Application of knowledge acquisition methods in a case-based reasoning tool

Alexandre José Carneiro Borges de Sousa

Dissertação de Mestrado

Orientador na FEUP: Prof. Eduardo Gil da Costa

Mestrado Integrado em Engenharia Industrial e Gestão

2018-01-22

A minha mãe, a minha irmã, a Guida

"Sempre chegamos ao sítio aonde nos esperam."
Livro dos Itinerários, José Saramago

Abstract

At Laborial, a construction company in the laboratories industry, the commercial team set the goal of being more market-responsive, i.e. reduce the lead-time of the first response to the client. However, it lacks the knowledge to elaborate commercial proposals, which is performed by technicians in a different department.

A case-based-reasoning tool was developed to address the problem of automatically elaborating a commercial proposal based on input specifications. A domain-independent technique to overcome the knowledge engineering effort during the adaptation phase of CBR is investigated. The goal is the acquisition of adaptation knowledge based on knowledge already acquired within the system, rather than hand-coding domain knowledge.

In terms of accuracy in product reference identification the overall system results are not significantly improved by using such a methodology. Concerning the adaptation mechanism alone, the system is competent in properly adapting a previous situation to the current problem requirements, for one product family (accuracy = 67%), and unable to adapt for the other product family (accuracy = 0%). Notwithstanding, concerning product cost estimation, the system is able to estimate the cost within the required range (±20%) and concerning the adaptation mechanism alone, for one product family cost estimates are within the range (±3%), whereas for the other product type it is in the range (±33%).

Despite the results it is argued that the investigated methodology for adaptation knowledge acquisition may be suitable for the task at hand, however there are some caveats. It is argued that the case-base size is constraining both the adaptation knowledge acquisition mechanism and also the overall system; nevertheless, it is expected that as the case-base grows, by acquiring new cases, the system may be able to become more competent, as well as the adaptation knowledge acquired may be refined. Aside from that, it is argued that poor product requirements definition may be constraining the results obtained for the product family with worst results.

Resumo

A Laborial é uma empresa de construção na indústria de laboratórios cuja equipa comercial definiu como objectivo aumentar a sua capacidade de resposta, nomeadamente conseguindo responder mais rápido aos pedidos de orçamentação de um cliente. Contudo, a equipa comercial não tem conhecimento suficiente para elaborar orçamentos uma vez que estes são elaborados por técnicos especializados de um departamento diferente.

Recorrendo à metodologia de raciocínio baseado em casos, desenvolveu-se uma ferramenta que permita elaborar orçamentos automaticamente. Foi investigada uma metodologia que permita a adaptação de casos passados, usando o mínimo conhecimento do domínio possível. O objectivo desta metodologia é possibilitar a aquisição de conhecimento que permita a adaptação de casos através do uso de conhecimento já adquirido, e no sistema, em vez de codificar o conhecimento de um técnico especializado.

Em termos de precisão na identificação da referência do produto os resultados globais do sistema não são significativamente melhorados pelo facto de esta metodologia ser usada; considerando apenas o mecanismo de adaptação, o sistema é competente na adaptação de casos anteriores para uma das famílias de produtos (precisão = 67%), e incapaz de adaptar qualquer caso para outra das famílias de produto (precisão = 0%). Em termos de estimativa de custo o sistema é capaz de estimar o custo dentro do intervalo pedido (±20%); considerando apenas o mecanismo de adaptação, o sistema é capaz de estimar o custo para uma das famílias de produto no intervalo ±3%, e para a outra adapta as soluções no intervalo ±30%.

Apesar dos resultados conclui-se que a metodologia de adaptação investigada pode ser adequada para a tarefa, mas há, contudo, dois pontos a considerar. A amostra disponível pode ter restringido tanto o sistema como o mecanismo de aquisição de conhecimento; contudo, é expectável que à medida que a base de casos aumenta o sistema se torne mais competente, e o sistema de adaptação consiga ser mais refinado. Além disso, é possível que a má definição das especificações possa ter condicionado o sistema na geração de resultados para a família de produtos que apresenta piores resultados.

Acknowledgements

I want to express my gratitude to professor Eduardo for his availability and guidance throughout the project development.

In the company, special thanks to Isabel for her help and care, to Ângela for being present when it was needed and care, and Pedro for his full availability, help and talks about computer science.

Special thanks to Joana for reviewing the text and for the invaluable contributions.

Contents

Nomenclature

CBR	Case-based reasoning
AI	Artificial Intelligence
CD	Commercial Department
PD	Projects Department
PM	Project Manager

List of Figures

List of Tables

1 Introduction

The present dissertation was developed in a company environment at "Laborial SA" (Laborial) in the scope of the Master Degree in Industrial Engineering and Management, at the Faculty of Engineering of the University of Porto.

1.1 Project within the company

This project was developed at Laborial, a company that offers solutions for laboratories. The project was integrated in the Commercial Department (CD) with the purpose of developing a tool that allows the automation of a project's commercial proposal based on preliminary specifications.

For organizational reasons the project manager (PM) integrates the CD – besides being responsible for the project within the company, it is also the commercial connection to the client. The PM is the only entity that is involved from the project opening to its closure, notwithstanding that in different phases the project leader is another entity from a different department.

Any project requires a commercial proposal that is developed in a different department – Projects Department (PD). The PM contacts the client to know his requirements and specifications, if any, that within the company is formalized by filling a standard input form. PD technicians have to interpret the input form data, and answer the client inquiry with the products (solution) that best match the customer requirements. As such the PM is in contact with the client but it is the PD technician that knows the product. In fact, it was possible to understand that the client, and even the PM, can ask for a technically infeasible solution, which is assessed by the PD technician. Not that the PM does not know the product, but technical details – such as structure integrity, chemical restrictions – are, usually, out of their scope.

Commercial proposal development is an iterative process between the client, PM and PD – usually, until the customer order more than one commercial proposal is developed. Although more than one commercial proposal is delivered to the client, usually previous proposals are used as a basis, and as such, the subsequent commercial proposals are a rework of the previous ones.

1.2 The Company

Laborial is a supplier of technical furniture and chemical fume cupboards, development of integrated projects, custom made and technological solutions for laboratories (Laborial Soluções para Laboratório 2016).

The company is certified on the management systems of Quality (according to ISO 9001), Research, Development and Innovation System (according to NP4457) and Environment (according to ISO 14001) (Laborial Soluções para Laboratório 2016).

As it offers highly customized solutions its activity is developed as project management. Every time the company receives an inquiry from a potential client, or it makes prospection, a new project is opened, and any project is allocated to a project manager, in this case, by geographical area. For the fact that each solution is potentially unique, its production system is make-to-order and inventory management is just in time.

1.3 Project Goals and Motivation

The main goal of the project is to reduce the lead-time of the first response to the client, i.e. increase the PM response power. As presented in detail in sub-chapter 3.3, the average lead-time of the first commercial proposal is 8 days and the main motivation is to have a tool that the PM can use as a first commercial proposal reply to the customer quicker. The scope of the tool are projects such that the standard product line satisfies the customer requirements; as requested by the CD, technical furniture and fume cupboards are under analysis. Whereas the first-reply lead-time was computed considering all projects the company develops (and not only projects involving technical furniture and fume cupboards), the value works as a benchmark for the potential such a tool represents for the company.

An automatic commercial proposal tool can address that, in the sense that, as of now, the PM response time is contingent on PD restrictions, as after collecting the customer requirements and formalizing them within the company, the responsibility falls to the PD. The goal is that such a tool automatically elaborates a commercial proposal that identifies the most adequate products and their cost, within an error margin of 20% on cost – the tool creates the proposal based on product's cost, to which commercial conditions, namely margin, are set by the PM. It is considered, by the CD, that 20% error on cost is manageable, considering commercial margin is yet to be added.

It is expected that such a tool can also reduce the lead-time of other activities within the PD as the department is no longer overloaded with standard projects, and needs to deal only with highly specialised projects. Moreover, a potential side effect is that for standard projects the output from the tool may ease the PD overload by providing a commercial proposal that might serve as a basis of work, i.e. in case the output is correct the PD technician needs only to validate it.

1.4 Methodology

A case-based reasoning (CBR) tool was designed to address the company problem of being more market responsive.

As discussed in section 2.2, there are several open areas that should be tackled in an efficient CBR system. This dissertation addresses, mainly, related problems to case adaptation, specifically, the open research question Kolodner (1992) put up in her early research - researchers "must address whether there is a general set of adaptation strategies that we can start with for any domain and that provide guidelines for defining specialized adaptations strategies". The importance of case adaptation lays in the possibility to broaden the coverage of past cases (Wilke et al. 1997).

Moreover, Wilke et al. (1997) argue that "when developing a CBR system a general aim should be to manually compile as little knowledge as possible but only as much as necessary", i.e., the CBR system should be based on knowledge-light approaches – the distinction between knowledge-light and knowledge-intensive is discussed in section 2.3.2.

There is some research on domain-independent strategies to overcome the knowledge engineering effort for the adaptation phase, (Wilke et al. 1997; Hanney and Keane 1997; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2008). In line with

research from these authors, this dissertation is an investigation in methods that allow the acquisition of adaptation knowledge, in a knowledge-light approach.

1.5 Dissertation Outline

The present document is organized in 5 chapters.

In this first chapter a short description of the project and its targets has been done, together with a presentation of the company where the project was held and the methodology followed.

In chapter 2 the theoretical framework is discussed – CBR methodology (sub-chapter 2.1) and issues (sub-chapter 2.2) that a CBR system must address within each phase; next (sub-chapter 2.3), as the object of study concerns the adaptation phase of a CBR system, the importance of this phase is discussed, and its issues are further explored, namely, how to acquire knowledge that enables case adaptation that does not lead to hand coding of specific domain knowledge, thus an *ad hoc* solution.

In chapter 3 the problem framework is presented – what is the current situation at the company, and how it fits the investigation into an automatic commercial proposal tool – first the data collected is characterized, as well as the pre-processing performed, if any (sub-chapter 3.1); second, the commercial proposal is described (sub-chapter 3.2) followed by the quantification of the first commercial proposal lead-time (sub-chapter 3.3), as the CD responsible feels the PM should be more responsive but it is not quantified how long he or she take to reply back to client; next (sub-chapter 3.4) a product characterization is performed – considering the goal of cost estimation, each product family weight on global return is assessed; last, considering the tool's perspective the problem spaces (sub-chapter 3.5) are presented, namely problem descriptions space (section 3.5.1) and the problem solutions space (section 3.5.2).

In chapter 4 the developed proposals are presented – the revised input form (sub-chapter 4.1) and the developed tool (sub-chapter 4.2). In the end the results and a brief discussion are presented (sub-chapter 4.3).

In chapter 5 the main conclusions are presented (sub-chapter 5.1) and some limitation of the tool are discussed, as well as some possible paths of improvement (sub-chapter 5.2).

2 Theoretical Framework

In this section the theoretical framework is discussed. It is structured as follows: first, the case-based reasoning (CBR) methodology is presented and the main challenges/open problems to its implementation are discussed. Following, it is shown the relevance of the automatic knowledge acquisition for the adaptation phase of the CBR methodology, and a methodology to achieve it is presented.

2.1 Case-based reasoning

CBR is a problem solving methodology in which, when faced with a new problem, the reasoner remembers previous situations similar to the current one and uses them to help solving the new problem, (Kolodner 1992). It "is a methodology for both reasoning and learning", (Kolodner 1992).

2.1.1 Context

Early CBR research dates back to 1982 with the work of Roger Schank based on cognitive sciences, namely, the research into how people remember information and, in turn, remind that information (Watson 1999; Leake 1996; de Mántaras et al. 2005).

Kolodner (1992) presents some examples of people's approach to problem solving by remembering previous situations: attorneys using cases as precedents for constructing and justifying arguments in a new case; a doctor diagnosing a patient by recalling a different patient he has seen previously, with similar symptoms; a car mechanic remembering other similar problems and considering their solutions to address the current problem, are some of the examples. The author adds that "if we watch the way people around us solve problems, we are likely to observe case-based reasoning in use all around us (…) In general, the second time solving some problem or doing some task is easier than the first because we remember and repeat the previous solution. We are more competent the second time because we remember our mistakes and go out of our way to avoid them" (Kolodner 1992). Leake (1996) adds, "CBR as a cognitive model is supported by studies of human reasoning which demonstrate reasoning from cases in a wide range of task contexts". CBR is an attempt to emulate this human behavior to solve problems.

Some authors argue there are two major types of case-based reasoners: interpretative and problem-solving, (Kolodner 1992; Leake 1996; de Mántaras and Plaza 1997). Given the purpose of this work, problem-solving CBR is under analysis so when CBR is mentioned it is meant problem-solving CBR.

2.1.2 CBR Overview

The process of solving a problem through CBR requires a problem description, measuring the *similarity* of the problem at hand to the previous problems stored in the *case base* (or *memory*), which contain their known solutions. One, or more, similar cases are *retrieved*, and

an attempt is made to *reuse* their solutions, possibly after *adapting* them to account for the differences between problem descriptions. The proposed solution is then *evaluated*, either by being applied to the initial problem or assessed by a domain expert. Following revision of the proposed solution, and in the light of its evaluation, the newly solved problem, i.e., the problem description and its solution, can be *retained* as a new case and the system has learned to solve a new problem (de Mántaras et al. 2005).

The classic CBR process proposed by Aamodt and Plaza (1994) is presented in Figure 1.

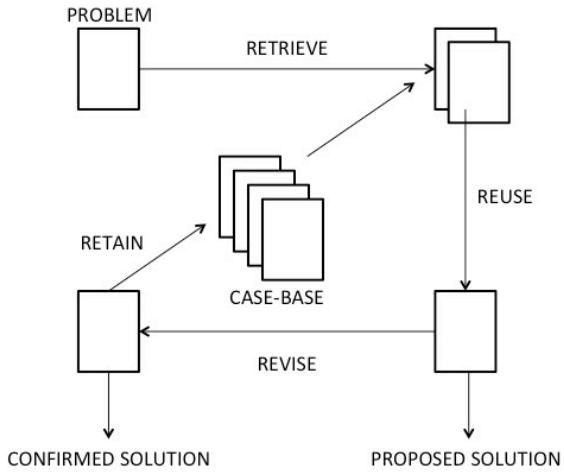

Figure 1 - CBR process (Adapted from Aamodt and Plaza 1994)

The authors coined the 4 Re's mnemonic: Retrieve, Reuse, Revise and Retain:

- "RETRIEVE the most similar case or cases;
- REUSE the information and knowledge in that case to solve the problem;
- REVISE the proposed solution;
- RETAIN the parts of this experience likely to be useful for future problem solving", (Aamodt and Plaza 1994).

The revise phase is the only of the 4 phases of the CBR methodology that is external to the system – the proposed solution is evaluated either by asking a teacher or by performing the task in the real world, (Aamodt and Plaza 1994; Kolodner 1992).

There are two implicit assumptions in CBR. First, similar problems have similar solutions; as such solutions to previously solved problem are a useful starting point in solving a new problem. Second, problems tend to recur; thus it is useful to remember past cases, in the sense they might be useful for future problem solving, (Leake 1996). Despite pointing out these assumptions, Leake (1996) argues that CBR systems can also be used in creative problem solving provided there is an appropriate structure of flexible case retrieval and/or adaptation.

Regarding the first assumption, Leake (1996) states that a CBR system explores two types of similarity, each related with a different space – the space of problem descriptions, and the space of problem solutions. Moreover, the two types of similarity explored relate to different phases in the CBR cycle. Figure 2 presents the system perspective, where the two types of similarity, interactions between the different spaces and phases, are depicted.

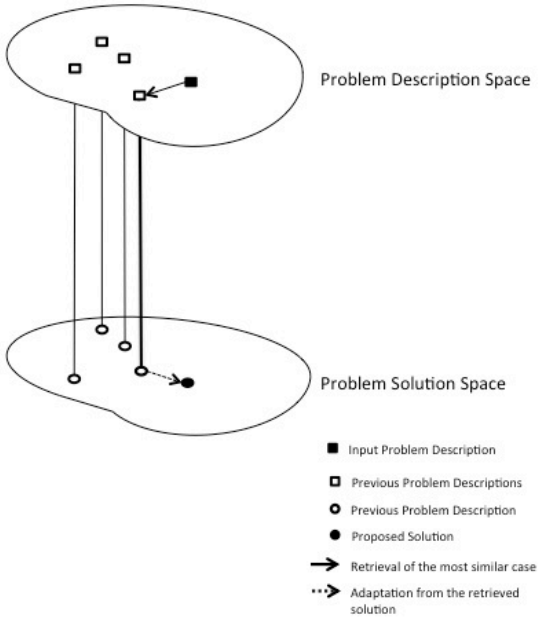

Figure 2 - CBR knowledge and problem spaces (Adapted from Leake 1996)

"When presented with a new problem, a CBR system (…) searches for problems with similar problem description", (Leake 1996). Similarity assessment is processed in the space of problem descriptions, during the retrieval phase. "The solutions of those problems are used as the starting point for generating a solution to the new problem. With the right way of describing the problem, similar problems will have solutions that are similar i.e., easy to adapt to the new situation", (Leake 1996). Solution adaptation is processed in the space of problem solutions during the reuse phase.

It is worth noting that, according to Watson (1999), "CBR describes a methodology for problem solving but does not prescribe any specific technology". In fact, up to this point, it was described what CBR is intended to do but not how it does it, technically speaking. As such, the same author proceeds, "a case-based reasoner can use any technology provided the system follows CBR's guiding principles", in fact, "CBR systems must use other technologies; since CBR has no technology to call its own per se", (Watson 1999).

This is, in fact, the object of study of this dissertation – which technologies to use, specifically for the adaptation of cases, which is a sub-process of the reuse phase, and is, also, "one of the major challenges during designing a CBR system", (Wilke et al. 1997; Policastro, Carvalho, and Delbem 2008).

2.1.3 CBR within the Artificial Intelligence field

According to Aamodt and Plaza (1994), "Case-based reasoning is a problem solving paradigm that in many respects is fundamentally different from other major AI approaches", in the sense that it does not rely solely on general knowledge of a problem domain, or associations, but uses specific knowledge of the previous situations experienced. "Reasoning is often modeled as a process that draws conclusions by chaining together generalized rules, starting from scratch. CBR takes a very different view. In CBR, the primary knowledge source are not generalized rules but a memory of stored cases recording specific prior episodes. In CBR new solutions are generated not by chaining but by retrieving the most relevant cases from memory and adapting them to fit new situations. Thus in CBR, reasoning is based on remembering", (Leake 1996). Moreover, "a second important difference is that CBR is an approach to incremental sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems",

(Aamodt and Plaza 1994). Kolodner (1992) also points that, "within AI, when one talks of learning, it usually means the learning of generalizations. While the memory of a case-based reasoner notices similarities between cases and can therefore notice when generalizations should be formed, inductive formation of generalizations is responsible for only some of the learning in a case-based reasoner". In a CBR system, learning occurs not only because the system induces generalizations based on similarities between cases but mostly by accumulation of new cases and the assignment of indices in the case memory, for later use, (Kolodner 1992; de Mántaras and Plaza 1997). Regarding case acquisition, new cases provide the system more contexts for solving problems; a reasoner whose cases cover more of the domain will perform better than one whose cases cover less of the domain and one whose cases cover successful and failure will perform better than one whose cases cover only successful situations, (Kolodner 1992).

2.2 Issues in CBR

As in other AI approaches, there is not a universal CBR method that suits every problem application, (Aamodt and Plaza 1994). As such, the key aspect in a CBR system is how it is implemented, namely, which technologies or methods are used to achieved the desired goals of each phase, (Watson 1999; Aamodt and Plaza 1994).

In this sub-chapter, the core problem areas of implementing a CBR system are discussed. Broadly speaking, the main problem areas to be tackled concern knowledge representation and the four Re's in (Aamodt and Plaza 1994)'s framework. Besides those issues, system maintenance is also discussed.

In spite of being presented separately, for the sake of clarity, it is noted that "the steps of CBR are not independent. Considering them together provides an advantage over studying them individually, because their relationships can be exploited to facilitate and constrain processing in each one", (Leake 1996).

2.2.1 Knowledge representation

Knowledge representation is a key issue in CBR, especially for the fact that the methods used in the 4 phases of CBR are designed according to it. The reasoner "is heavily dependent on the structure and content of its collection of cases" (Aamodt and Plaza 1994), but case structure and the case base contain only part of the knowledge in the system.

In his work from 1995, Richter identified four knowledge containers in a CBR system:

- "The *vocabulary* used to describe the domain;
- The *case base*;
- The *similarity measure* used for retrieval;
- The *solution transformation* used during adaptation" (quoted in Wilke et al. 1997).

Stored knowledge may include domain knowledge as well as problem solving knowledge. In designing a CBR system the developer decides how the knowledge is distributed to the different containers, based on available knowledge and engineering effort. A decision is made as: if there is not enough knowledge available to fill one container, or if it is expensive to acquire, there needs to be a knowledge transformation from some containers to others, (Wilke et al. 1997). Moreover, knowledge within containers is not invariant, i.e. it can be shifted between containers either manually or modelled using a learning process (Richter and Aamodt 2005).

According to Aamodt and Plaza (1994), "the representation problem in CBR is primarily the problem of deciding what to store in a case, finding an appropriate structure for describing case contents, and deciding how the case memory should be organized and indexed for

effective retrieval and reuse". One can conclude that all but the revise phase are directly affected by how knowledge is represented in CBR.

Concerning the representation related to the vocabulary container, Bergmann, Kolodner and Plaza (2005) argue that three major types of case representation are the most widely used – feature-vector, structured and textual (or semi-structured). Besides these, the authors argue that there are some specialized representations of cases for specific tasks, such as plans in the planning task. Feature-vector represents a case as a vector of attribute-value pairs; structured represents cases "as clusters of relations between the kinds of elementary objects that comprise them"; textual imposes a weak structure on the cases – it is mainly related with systems dealing with text documents such as bug reports or FAQ (Bergmann, Kolodner, and Plaza 2005).

It is important to notice that the way the vocabulary container and the similarity measure container designs are not independent – "representation of cases and the way similarity is assessed during retrieval are strongly related to each other. Distance-based similarity metrics are easy to apply to feature vectors, while techniques related to information retrieval can easily be applied to textual representations. Frame-based cases often require knowledge-intensive indexing and matching algorithms" (Bergmann, Kolodner, and Plaza 2005).

Concerning the case base container, namely how to index the cases, it affects not only the retrieval of the most meaningful cases but also an efficient search in the case base (Richter and Aamodt 2005). Efficiency can become a problem if the number of cases is large - "When the case memory is large, a good organization of the memory is a must because a simple linear organization, like a list, is very inefficient for retrieval. A hierarchical organization is necessary" (de Mántaras and Plaza 1997).

2.2.2 Retrieval

At the beginning of this chapter, it is stated that the reasoner remembers previous situations similar to the current one – by remembering it is actually meant that the system is able to recall relevant cases. That is the goal of the retrieval phase - to retrieve the most meaningful case, or set of cases, to solve the problem.

To start, there is the need to define some measure that assesses what "meaningful" is in the context of the problem. The approach to define the similarity metric can be classified as knowledge-poor or knowledge-intensive approach. The distinction concerns explicitly represented domain knowledge – cases contain explicit knowledge, but in this case it is denoted as specialized (or specific) domain knowledge (Aamodt and Plaza 1994). Comparing a problem might simply involve noticing its input description (knowledge-poor approach), or it might involve making an effort to "understand" the problem within its context, i.e., not only comparing the input descriptors but also interpreting the values they take (Aamodt and Plaza 1994). "To understand a problem involves to filter out noisy problem descriptors, to infer other relevant problem features, to check whether the feature values make sense within the context, to generate expectations to other features, etc." (Aamodt and Plaza 1994). de Mántaras and Plaza (1997) argue that understanding the problem context might be crucial as "an additional difficulty is that the similarity metrics must take into account that not all features have the same importance. While it would seem that some sort of weighted similarity measures could do, in fact this is not always possible because the importance of some features is context dependent". Regarding knowledge-poor approaches, Kolodner (1992) warns that "one problem is that sometimes two cases need to be judged similar even though they share few surface features (…) what they have in common is more abstract features". Nevertheless, a knowledge-poor approach to similarity assessment has its advantages in some domains, (Aamodt and Plaza 1994; de Mántaras et al. 2005), namely domains where general domain

knowledge is difficult, or even impossible, to acquire (Aamodt and Plaza 1994). On the other hand, if domain knowledge is easy to acquire a knowledge-intensive should be favoured, (Aamodt and Plaza 1994), as, even though computationally expensive, it has the advantage that more relevant cases are retrieved (de Mántaras et al. 2005). In a compromise approach, de Mántaras et al. (2005) suggest that a way of mitigating the extra cost of a knowledge-rich approach is to combine knowledge-poor and knowledge-intensive approach in a two step-stage retrieval, as presented by Borner in 1993, where a fast retrieval of candidate cases, using a knowledge-poor approach, is followed by a more expensive assessment of cases' similarity, using a knowledge-intensive approach, (quoted in de Mántaras et al. 2005).

Addressing these problems comprises what is "called the indexing problem. Broadly, the indexing problem is the problem of retrieving applicable cases at appropriate times" (Kolodner 1992), despite the problems presented. "In general, it has been addressed as a problem of assigning labels, called indexes, to cases that designate under what conditions each case can be used to make useful inferences. These labels have been treated much like indexes in a book" (Kolodner 1992). de Mántaras et al. (2005) state that to "help assure that useful cases are retrieved without extensive computation is to develop carefully crafted indexing vocabularies to describe cases, so that the explicit description of a case captures the features that determine its relevance".

Notwithstanding similarity importance for a CBR system and what was presented so far, "there is a growing awareness of the limitations of similarity-based retrieval" and "several authors have questioned the basic assumption on which similarity-based retrieval is based, namely that the most similar cases are the most useful for solving the target problem", (de Mántaras et al. 2005), i.e., effective retrieval does not only concern the most similar case, but cases that are the most *usefully* similar (Leake 1996; de Mántaras et al. 2005). In a research pointing in this direction, a methodology – adaptation-guided retrieval – is developed, where the similarity metric is "augmented" to take into account the adaptiveness of the retrieved case, as the authors argue "it is often unwarranted to assume that the most similar case is also the most appropriate from a reuse perspective" (Smyth and Keane 1998). Besides considering adaptation-guided retrieval, de Mántaras et al. (2005) also point to other works , which extend retrieval considering other aspects, namely "diversity-conscious retrieval", to avoid the potential trap that the most similar cases are very similar to each other, resulting in limited choice, and "compromise-driven retrieval" that considers the compromises the user may be prepared to accept.

Another important aspect to consider is how much effort should be spent in this phase considering the next phase concerns adaptation of the retrieved solution (Kolodner 1992). Not implying the retrieval phase is less relevant, because the proposed solution is going to be adapted, but, instead, the design of one has to take into account the other.

In the end of this phase, " relevant portions of the cases selected during retrieval are extracted to form a ballpark solution to the new case" (Kolodner 1992).

The issues of retrieval concern how to assess cases similarity and its relevance is related to the fact that the "remaining operations of adaptation and evaluation will succeed only if the past cases are the relevant ones" (de Mántaras and Plaza 1997).

2.2.3 Reuse

Whereas the retrieval phase concerns similarity between cases, the reuse phase is primarily concerned with dissimilarities between them. Assuming a new case does not exactly match an old case in the case base (i.e., similarity metric $\neq 1$), or the retrieval function does not find it, the retrieved case is a partial match to the new case, to be solved. The goal in the reuse phase is to identify meaningful differences between the retrieved cases and the new problem and,

based on that, understand how the solutions used in the previous case can be adapted to the context of the new problem.

Aamodt and Plaza (1994) point the two focal aspects: "(a) the differences among the past and the current case and (b) what part of the retrieved case can be transferred to the new case".

Reuse can be as simple as retrieving the solution of the most similar case; however, if there are significant differences between the problems descriptions, the retrieved solution may need to be adapted, in order to account for these differences (de Mántaras et al. 2005).

As Kolodner presents in her work from 1993 (quoted in Craw, Wiratunga, and Rowe 2006), there are three types of adaptation:

- *Substitution*: replaces the value of the retrieved solution, according to the new problem differences;
- *Transformation*: alters the retrieved solution by adding, or deleting components, according to the new problem differences;
- *Special*: Methods apply specialized heuristic knowledge to repair the retrieved solution, or replay the methods used to derive the retrieved solutions.

Substitution adaptation does not take into account how a problem is solved, it rather focuses on the equivalence of the solutions; Transformation adaptation uses information regarding how the retrieved problems were solved, and uses that method in the new context, rather than the solution itself (Aamodt and Plaza 1994). Transformation adaptation requires a richer case representation, as its representation must hold information regarding the problem solving method used, aside from the solution (Aamodt and Plaza 1994). It is worth noting that the types of substitution are presented in ascending order of domain knowledge requirements.

A general strategy to overcome the adaptation problem is to "come up with a set of adaptation strategies or heuristics. We can implement those and create a working system. This is rather *ad hoc*, however" (Kolodner 1992). Some authors argue that the use of hand-coded adaptation rules is the most widely used adaptation strategy (Hanney and Keane 1996, 1997; Policastro, Carvalho, and Delbem 2008). As Kolodner (1992) states early in her research, CBR researchers "must address whether there is a general set of adaptation strategies that we can start with for any domain and that provide guidelines for defining specialized adaptations strategies".

Some researchers addressed questions intimately related with this research direction proposed by Kolodner (1992), namely: how to acquire adaptation knowledge that is domain-independent – thus avoiding *ad hoc* tools, and rather applicable to any domain. The knowledge acquisition problem in the context of adaptation is further explored in section 2.3.

The relevance of this phase is that "adaptation provides a fundamental role in the flexibility of problem-solving CBR systems" (Leake 1996), as by being able to adapt the systems broadens the coverage of the description and solution spaces.

2.2.4 Revision

While the first two phases concern problem solving, the last two concern learning. When the proposed problem solution is not correct, an opportunity for learning from failure arises (Aamodt and Plaza 1994); "Failures reveal that learning is needed" and "help focus decisions about what to learn: the needed learning must help avoid future failures" (Leake 1996). To evaluate the proposed solution the system requires feedback.

This is a step usually performed outside the CBR system – the proposed solution is evaluated either by asking a teacher or by performing the task in the real world (Aamodt and Plaza 1994; Kolodner 1992). The result of this action provides the system feedback – the proposal

was successful or not. Kolodner (1992) considers this as one of the most important steps in a CBR system as collecting feedback enables the system to evaluate its proposals, which allows it to learn.

The learning process is contingent on the results – if the system provides the correct answer, there is no need for analysis; however, if the proposed solution is not correct, further analysis is required, in order to assess the differences between the proposal and the actual expected result (Aamodt and Plaza 1994; Kolodner 1992). "It involves detecting the errors of the current solution and retrieving or generating explanations for them" (Aamodt and Plaza 1994).

If a problem is solved successfully, the resulting solution is available for future reuse. However if it is unsuccessful, the system refines the initial domain knowledge (by revising indexing criteria, for instance) which allows the system to favour solutions that are more likely to be successful, based on this experience; it might simply store information about the failure to be analysed in the future, when more data is available; or as to provide a warning about possible future failures that should be avoided (Leake 1996). A CBR system might be able to learn both from task failures, i.e. the system provided the wrong solution, but also from expectation failure, i.e. the observed outcome is different from prediction (either for better or for worse), (Leake 1996).

Leake (1996) argues that an alternative to evaluation is to perform evaluation based on cases within the case library; given an adapted solution, similar cases are retrieved from the case base and provide a dynamic benchmark for judging the quality of the adaptation.

2.2.5 Retain

This phase, also considered a learning step for the system, consists in storing the new case – problem description and proposed solution (Aamodt and Plaza 1994). Kolodner (1992) also points out that, besides the case, there can be stored "any underlying facts and supporting reasoning that the system knows how to make use of, and its outcome", i.e., the reasoning performed in the problem solving process (de Mántaras et al. 2005). The goal is that in the end of the retain phase the system evolved, in the sense that it can solve, at least, one new problem – for a future problem that is equal, it has the solution, but it might also be able to adapt this solution to solve future problems that, despite not being equal, are similar.

de Mántaras and Plaza (1997) propose that only those cases that led to classification errors should be retained. In that case, Leake (1996) argues, information about the failure should also be stored, as it can provide a warning about possible future failures.

To perform this task it is necessary to decide "which information from the case to retain, in what form to retain it, how to index the case for later retrieval from similar problems, and how to integrate the new case in the memory structure" (Aamodt and Plaza 1994). Kolodner (1992) argues that in this phase the system must make the choice of how to index the new case in memory – "Indexes must be chosen such that the new case can be recalled during later reasoning at times when it can be most helpful. It should not be over-indexed, since we would not want it recalled indiscriminately".

The issue is associated with the fact that the case is stored with the expectation that it might be needed in the future. As such, the system must be able to anticipate the importance of the case to later reasoning, and index it in the memory accordingly. This also implies that memory's indexing structure and organization are updated in this step (Kolodner 1992). Indexing is not static and must evolve with the system as "new indices allow a reasoner to fine-tune its recall apparatus so that it remembers cases at more appropriate times" (Kolodner

1992). New indices must be created, or existing ones updated, given that "indices needed are inextricably tied to the contents of the case library (which may change)" (Leake 1996).

Besides the issues discussed, retain also shares the issues of retrieval (Kolodner 1992), which is due to the fact that cases are retained in the expectation that will be useful in the future, and as such, will eventually be retrieved.

2.2.6 Maintenance

CBR system's maintenance aims to tackle performance issues that the system might incur. Whereas early systems simply stored every new generated case, it soon became clear that questions such as the effect of design decisions about maximum size of case library, as well as how to decide which cases must be stored in order that sufficient coverage is provided, had to be tackled (Leake 1996).

The case-base should not grow uncontrollably as firstly according to Kolodner in 1996, large case bases are not necessarily required by CBR as the size depends strongly on the task being addressed (quoted in Leake 1996), and secondly there is a pitfall in letting the case base grow uncontrolled as it leads to degradation of system's performance, which results from increased cost of accessing memory (de Mántaras and Plaza 1997; de Mántaras et al. 2005) – this is called the utility problem in CBR (de Mántaras et al. 2005).

"Cases correspond to a form of speed-up knowledge in the sense that retrieval and reuse of similar cases are expected to provide more efficient problem solving than first-principle methods, with the additional cases increasing the range of problems that can be solved rapidly" (de Mántaras et al. 2005). This naïve approach fails to acknowledge the increased cost of accessing memory, which is caused by two conflicting consequences of case-base growth: average savings in adaptation effort tend to increase efficiency of the system whereas the average retrieval time tends to decrease the overall system efficiency (de Mántaras et al. 2005). Smyth and Cunningham have demonstrated in 1996 that as a result of case learning (case-base growth) the mean retrieval time tends to degrade while mean adaptation time tends to improve, but at an ever decreasing rate (quoted in de Mántaras et al. 2005); this occurs because initially each newly learned case is more likely to have a significant impact on adaptation as it is more likely to improve description and solution spaces coverage, however, when the case base is large, new cases are more likely to overlap with existing cases and, as a consequence, less likely to improve coverage, thus minimally impact on adaptation savings – eventually the increase in retrieval time as a result of a new learned case is higher than the adaptation savings offered (de Mántaras et al. 2005).

One way to address this issue, considering that the system searches the case base every time it solves the problem, is to control the case base size – both by restricting its growth in retain, and also by occasionally deleting stored cases (de Mántaras and Plaza 1997). "Surprisingly enough, in many speed-up learners even the apparently naïve random deletion of knowledge items (to maintain the knowledge base to some predefined size) works quite well for optimizing efficiency". However the same type of strategies do not translate directly to case-based reasoners (de Mántaras et al. 2005). "The problem stems from the fact that many case-based reasoners are not simply using case knowledge as a form of speed-up knowledge. Instead, cases are often a primary source of problem solving knowledge. Without cases, certain problems cannot be solved and thus the act of deleting cases may irrevocably reduce the competence of the system to solve new problems" (de Mántaras et al. 2005). Leake and Wilson propose, in 2000, that both competence and performance must be considered during maintenance, and argue the need for more fine-grained performance metrics that consider competing factors such as case base size, coverage and adaptation performance (quoted in de Mántaras et al. 2005).

Besides controlling the size of the case base, its structure also weighs in the system's performance. As presented in section 2.2.5, Kolodner (1992) proposes reviewing the memory structure when adding new cases to the case base.

2.3 Adaptation Knowledge Acquisition

The adaptation problem can be summarized as: "Although CBR systems avoid reasoning from first principles by remembering and reusing past solutions, substitution and transformation adaptation of retrieved solutions is often achieved by reasoning about how the problem differences should be reflected in the adaptation to the proposed solution" (de Mántaras et al. 2005). Addressing adaptation, specifically what to adapt, how to adapt and how to control the adaptation process, requires considerable domain knowledge which raises the question of how to acquire that knowledge (Leake 1996), as "the acquisition of adaptation knowledge can require a substantial knowledge engineering effort" (de Mántaras et al. 2005). Many systems have that knowledge encoded *a priori* into rule-based systems; however, "this approach raises the same types of knowledge acquisition issues that CBR was aimed at avoiding" (Leake 1996). "The difficulty of acquiring adaptation knowledge was identified in early CBR research but, until recently, relatively little effort has been devoted to automating the acquisition of adaptation knowledge" (de Mántaras et al. 2005).

Modelling appropriate adaptation knowledge is an issue in a CBR system. Unlike cases, adaptation knowledge is not readily available and might be hard to acquire (Wilke et al. 1997). Most CBR systems make use of hand coded adaptation rules (Hanney and Keane 1996, 1997; Policastro, Carvalho, and Delbem 2008), which tend to demand a significant knowledge engineering effort and be domain and application specific (Policastro, Carvalho, and Delbem 2008), "hence, the knowledge-engineering effort expended in one domain tends not to be reusable in other domains" (Hanney and Keane 1997). Mitra and Basak (2005) argue that, for the fact that solutions to case adaptation remain highly domain dependent (requiring a detailed problem specific knowledge), it leads to difficulty "to provide a general guideline for formulating case adaptation rules".

Mitra and Basak (2005) present a survey where a review of several research papers leads to the classification of different approaches to methods of case adaptation - see sub-section 2.3.1.

Wilke et al. (1997) propose a framework for knowledge acquisition methods, based on knowledge-light approaches, which is presented in sub-section 2.3.2.

Hanney and Keane (1996) propose an adaptation knowledge acquisition method based on learning from cases already acquired, i.e. the case base; their methodology is applied in a system of properties evaluation, with randomly generated cases. The methodology and issues to its application are presented in sub-section 2.3.3.

In a series of papers (Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006), which in some sense try to extend Hanney and Keane (1996)'s knowledge acquisition method based on the case base, not only do the authors create a rule set adaptation base but also explore machine learning algorithms to learn and generalize the adaptation knowledge; their methodology is applied in the tablet formulation design, in the pharmaceutical industry. Policastro, Carvalho and Delbem (2006, 2008) also explore the knowledge acquisition method based on the case base proposed by Hanney and Keane (1996), and extended it in a similar fashion to the works presented by Wiratunga, Craw and Rowe (2002; 2006); different machine learning methods are tested to learn and generalize the adaptation knowledge acquired; in their case, the proposed methods are tested in publicly available databases

accessible in UCI Meachine Learning Repository. These works are explored in sub-section 2.3.4.

2.3.1 Classification of Case Adaptation methods

Mitra and Basak (2005) classify case adaptation methods based on different aspects – domain knowledge requirement, adaptive capabilities of the adaptation model and type of adaptation knowledge.

Regarding domain knowledge requirement, the two-split division proposed is "knowledge-lean" and "knowledge-intensive" methods. Whereas the design of knowledge-lean adaptation algorithms is, ideally, domain knowledge independent or requires very little domain knowledge to perform the adaptation, knowledge-intensive methods require deep domain knowledge (Mitra and Basak 2005). Domain knowledge independence means the algorithm does not presume knowledge acquisition before learning, i.e. hand coded domain knowledge, rather they use already acquired knowledge inside the system to learn adaptation knowledge (Wilke et al. 1997) – in the cases presented in (Hanney and Keane 1996, 1997; Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008) the knowledge source within the system is the case base – it is noted, however, that, as Wilke et al. (1997) argue, a knowledge-light approach can use any knowledge container inside the system to acquire adaptation knowledge, not only the case base.

Considering the adaptive capabilities, the authors are referring to learning and three categories are proposed – adaptive, non-adaptive and implementation dependent. Non-adaptive are methods that perform case adaptation by static rules, and no learning mechanism is incorporated. Adaptive are methods that use some machine learning techniques to learn the adaptation knowledge. Implementation dependent are methods that the authors consider can be implemented either in adaptive or non-adaptive ways (Mitra and Basak 2005).

Concerning the type of adaptation knowledge, the authors are referring to the knowledge acquisition process, and divide it in two categories – static and dynamic. "Static adaptation knowledge is fed into the system initially by an expert and is not changed or modified after that" whereas "dynamic knowledge can be changed or modified from the experience of adaptation" (Mitra and Basak 2005).

A brief description of the research further discussed in the next sections is done, according to the terminology presented by Mitra and Basak (2005), i.e. domain knowledge requirement, adaptive capabilities of adaptation and type of adaptation knowledge.

Wilke et al. (1997)'s framework, which structures Hanney and Keane (1996, 1997)'s methodology, is a knowledge-lean approach, with regards to domain knowledge requirement.

Considering adaptive capabilities, Hanney and Keane (1996, 1997)'s methodology, and researches based on that (Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008), are adaptive. The way in which Wiratunga, Craw and Rowe (2002; 2006) and Policastro, Cravalho and Delbem (2006, 2008) try to extend the methodology proposed by Hanney and Keane (1996, 1997) is by incorporating machine learning techniques to learn and generalize from the adaptation knowledge acquired, whereas Hanney and Keane (1996, 1997) use an inductive algorithm to generalize the acquired rules.

Regarding the type of adaptation knowledge, Policastro, Carvalho and Delbem (2006, 2008)'s approach is static – the authors claim their approach assumes the CB is representative, i.e. "all future problems in the current domain are covered by the CB", which must be guaranteed during case acquisition and case base construction, and "therefore no retraining of the

adaptation mechanism is required". The other authors do not make it clear with regards to the type of adaptation knowledge.

2.3.2 A Framework for a Knowledge Light approach for Adaptation Knowledge Acquisition

The proposed approach to overcome the knowledge engineering effort is the automatic learning of adaptation knowledge (Wilke et al. 1997).

A knowledge-light approach for learning adaptation knowledge is one in which knowledge is transferred from other knowledge containers to the adaptation container (Wilke et al. 1997), i.e., it focuses on knowledge-transfer from other containers (already acquired knowledge), rather than acquiring and coding new knowledge.

The approach proposed consists in transferring knowledge that is transformed in adaptation knowledge by a learning algorithm, specifically inductive algorithms "because they generate general knowledge from examples" (Wilke et al. 1997). Figure 3 depicts this approach.

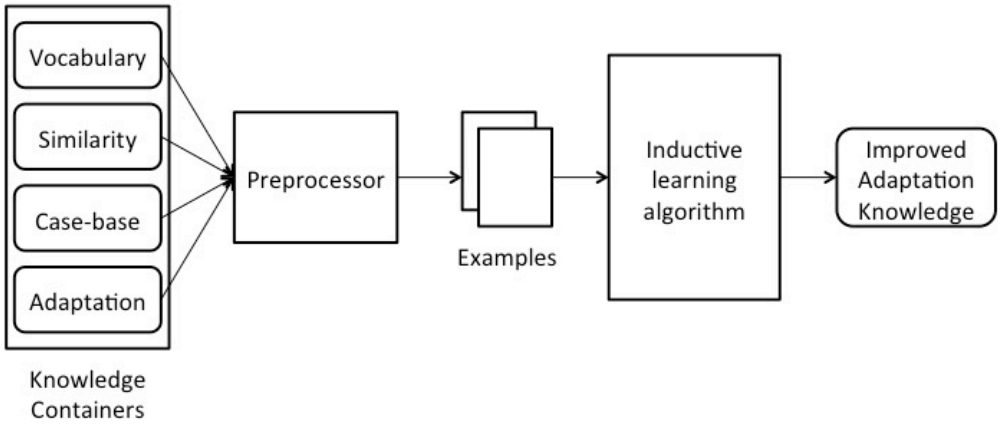

Figure 3 - A knowledge light framework for case adaptation (Adapted from Wilke et al. 1997)

Data is selected from different containers and, depending on the learning technique used, it is pre-processed into a suitable representation. The output of the learning process is integrated into the system, namely the adaptation container, in the form of adaptation knowledge. The integration step might require newly acquired knowledge to be combined with the already acquired knowledge such that, in the end, the adaptation container is updated with improved knowledge (Wilke et al. 1997).

Issues for designing knowledge light learning algorithms

The implicit assumption is that knowledge in the containers chosen to transfer knowledge from must be sufficient to solve the learning task as the learning goal and available knowledge constrain the inductive algorithms that might be useful for the learning task (Wilke et al. 1997).

Moreover, the designer has to decide how to measure the quality of knowledge for learning, handle noisy or unknown data and derive additional knowledge from containers to the learning algorithm. Aside from the knowledge selection task, a conflict resolution strategy might be required in case different sources of knowledge from different containers are contradictory – furthermore, a conflict resolution strategy is crucial for knowledge integration

in the final step, as it must be decided what happens in case contradictory knowledge arises, (Wilke et al. 1997).

Regarding the learning algorithm, the designer must assess its appropriateness for the learning task and available examples, as well as quantifying the required sample size, i.e. how many examples the learning algorithm needs (Wilke et al. 1997).

Despite contending for the use of knowledge-light approaches, the authors argue that it might be not always appropriate, namely for complex adaptation knowledge acquisition (Wilke et al. 1997).

2.3.3 Methodology for Learning Adaptation Knowledge From the Case Base

In a pioneering work, Hanney and Keane (1996) propose a methodology for learning adaptation knowledge from the case-base – using (Wilke et al. 1997)'s terminology, a knowledge-light approach for transferring knowledge from the case base container to the adaptation container.

The case base contains a lot of implicit domain knowledge, as such it makes sense trying to learn that knowledge in adaptation rules (Hanney and Keane 1997). In (Hanney and Keane 1996) the authors argue that it should be feasible to compute all feature-differences between cases and examine how these lead to differences in the case solution. From that analysis it should be possible to automatically learn a set of adaptation rules. "The implicit assumption here is that differences that occur between cases in the case-base are representative of differences that will occur between future problems and the case-base" (Hanney and Keane 1996, 1997).

The adaptation-rule learning algorithm has two major components - adaptation-rule generation and rule application. To generate adaptation rules a pair-wise comparison of cases is performed; feature differences are noted and become the antecedent part of an adaptation rule, with the consequent part of the rule being the differences noted between the compared cases solutions. A generalisation of the rule set generated is performed for rules with matching antecedents (Hanney and Keane 1997). The rule application part consists in the application of rules that best deal with the feature differences found between a target problem and the retrieved cases (Hanney and Keane 1997).

The rules created are represented by seven properties – "rule identifier, features, values, context, solution-change, gen and confidence-rating. The 'features' slot contains those features with different values in the case pair. The 'values' slot contains the value pairs of the features in the features slot. The 'context' slot contains those feature values that the case pair share that have an impact on solution change. The 'solution change' is the rule consequent and is an expression of the change in the solution. In the case of numeric solutions, it could be simply additive. For symbolic solutions, it is a transition from one class to another class. The 'gen' slot notes whether the rule was generalised", and the 'confidence-rating' is used to denote a measure of the confidence in a rule, based on its frequency (Hanney and Keane 1996). Rules' frequency is stored in a different set, but each rule has an associated frequency (Hanney and Keane 1996).

Issues in Adaptation Rule Learning from cases

The first raised question to this method is how to choose cases from the case base for comparison, in order to ensure the adaptation rules generated are meaningful. Moreover, after the generating procedure, processing might have to be carried out to determine which of the rules thus generated are actually useful for the system (Hanney and Keane 1996).

Another point to take into consideration is, as previously discussed, there should be a relation between retrieval and adaptation, i.e., it has to be addressed how to maximise the cooperation between retrieval and adaptation (Hanney and Keane 1997).

Furthermore, the rule generation process described creates the potential for the generation of a rule set several times larger than the original case-base. If each case is compared with only one other case, each comparison leads to one adaptation rule, thus the rule set created has the same size as the original case base; however, it is not uncommon for the retrieval phase to retrieve more than one case. There is a need to consider how to refine the large rule sets extracted from the case base as "comparing every case in the case-base with every other" is a strategy that "soon gives rise to an unmanageable prolixity in rule generation. It makes more sense to minimise the generation process by only considering some cases for comparison" (Hanney and Keane 1997).

Moreover, despite their proposal for automatic knowledge learning, Hanney and Keane (1997) argue that one should consider the "use of domain knowledge to guide adaptation rule learning", as "there are known regularities in the domain which are not evident from an examination of the case base" (Hanney and Keane 1996). The authors classify the four types of domain knowledge in the following way: *known adaptation rules* "are adaptation rules already known by the system designer"; *irrelevant features* "are features that have no effect on the solution"; *contextual dependencies* are relations between feature values and the solution such that "the solution change arising from a feature value change depend on the context"; and *feature interaction* deals with the knowledge of the effect of some features, which may be interacting with other features, or non-interacting (Hanney and Keane 1996). The authors argue domain knowledge should constrain the rule generation process in order to avoid unproductive case comparison since it might produce adaptation rules already known, and thus no real learning occurs (Hanney and Keane 1996).

In addition to rule generation, the authors argue that domain knowledge is a useful guide for rule application - known irrelevant features can be used to reduce the list of differences between target and retrieved case; contextual dependency can aid in assessing rule applicability by checking context; and overall the adaptation rule application process can be boosted by reduction of differences, if domain knowledge is applied prior to adaptation rules, i.e. the use of domain knowledge can help prevent incorrect adaptation by allowing the system to enforce constraints on adaptation rule applicability" (Hanney and Keane 1996).

The last point to be addressed is how to choose the best adaptation rule when there is more than one adaptation rule satisfying the feature differences between a target problem and the retrieved cases (Hanney and Keane 1997). The approach the authors presented is to assign to each rule a "confidence rating that reflects the number of duplicates it captures (the higher the rating the more duplicates it captures)" (Hanney and Keane 1996). The implicit assumption is that the more a given set of feature differences and consequent solution change occur (the higher the frequency and thus the higher the confidence rating) the more the system designer can be confident that this rule is likely to be the most appropriate for the situation at hand.

Hanney and Keane (1996)'s approach for a knowledge-light approach to adaptation rule learning is an attempt to "improve on the performance of a retrieval system without costly knowledge engineering". Despite being a knowledge-light approach, the authors argue for the use of some domain knowledge to guide the knowledge acquisition process, as well as rule application.

2.3.4 Use of machine learning techniques to learn and generalize adaptation knowledge

Hanney and Keane (1996) state that "generalisation has an important role to play in filling gaps in the rule-set; it allows a general pattern to be induced from a set of specific rules, producing a wider coverage difference"; the "main advantage of generalisation is that it broadens the applicability of the specific adaptation rules. The system induces a more general adaptation rule given several similar but non-identical, specific rules" (Hanney and Keane 1996). The generalization the authors suggest is "rule consequents can be generalised by taking the average, for numeric solutions, and the most common class change for symbolic solutions". Some authors (Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008), follow Hanney and Keane (1996, 1997)'s proposed framework to generate adaptation rules but differ in the process of rule generalization, by applying learning algorithms more sophisticated than simply averaging numeric solutions, or the most common class for symbolic solutions.

In a similar fashion to Hanney and Keane (1996, 1997), Wiratunga, Craw and Rowe (2002; 2006) and Policastro, Carvalho and Delbem (2006, 2008) perform a pairwise comparison of cases in the case base which is used to create an auxiliary case-base. Figure 4 depicts the approach.

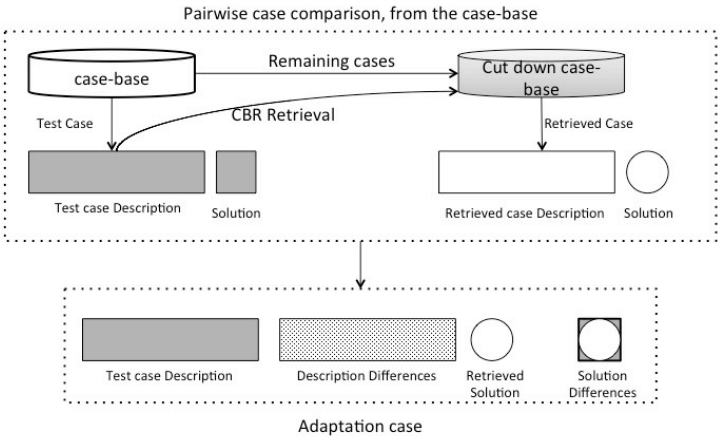

Figure 4 - Adaptation case-base generation (Adapted from Craw, Wiratunga, and Rowe 2006)

The adaptation-case-base is generated as follows: a leave-one-out approach is performed for all cases in the case-base; for all cases, each case is set as the test case and the remaining cases comprise a cut down case base from which a CBR retrieval is performed. For each retrieved case, features description is assessed and features differences and solution change is noted. The adaptation case thus generated is composed by four components – test case description, differences between test case and retrieved case descriptions, retrieved solution and solution change. The rule antecedent is the test description and description differences, and the rule consequent, which is the retrieved solution and solution differences.

By following this approach, the adaptation rules created are generated according to the structure in which they are to be applied – when a new case enters the system retrieval is performed and, for the most similar cases, differences are assessed and rules covering those differences are used to adapt the retrieved solution.

Whereas in (Hanney and Keane 1996, 1997; Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008) the approach used to generate adaptation rules is the one just described, based on Hanney and Keane (1996, 1997)'s methodology, they diverge in the way rule generalization is approached.

Wiratunga, Craw and Rowe (2002) use a RISE algorithm and boosted version of RISE, and both are compared. Craw, Wiratunga and Rowe (2006), the same authors, explore the use of a decision tree algorithm (C4.5) and compare it with RISE, developed in a previous work (Wiratunga, Craw, and Rowe 2002). Policastro, Carvalho and Delbem (2008) experiment different learning algorithms, namely Neural Networks (the multi layer perceptron) (MLP-NN), Support Vector Machines (SVM) and a tree learning algorithm (M5). Aside from using learning algorithms, the authors experiment with the use of a learning committee – a learning algorithm takes as input the output of the three learning algorithms and is responsible for the final decision. In their case, the same types of learning algorithms are tested, i.e., committee of MLP-NN, a committee of SVM and a committee of M5. Each committee takes as input the same output, i.e., the output of the learning algorithms MLP-NN, SVM and M5.

Whereas the results by different authors cannot be directly compared between them, as different problem domains were used, it is possible to understand the impact of their approaches for their domains, in terms of adaptation.

3 Problem Framework

In this chapter the problem presented by the company is described. It is structured in the following way: in sub-chapter 3.1 the data collection process is described; in sub-chapter 3.2 the commercial proposal is characterized; in sub-chapter 3.3 the lead-time of the first commercial proposal is quantified; in sub-chapter 3.4 the product is characterized, by product families; and in sub-chapter 3.5 the problem space characterization, from the perspective of the tool developed, is characterized.

3.1 Collecting Data

In the subsequent sections a description of data collected is made. As different types of data were collected, with different goals, it is organized considering the goal according to which data was collected.

3.1.1 Data to characterize commercial proposals

Commercial proposals characterization was made based on data collected from the information system. The sample size was 1023 records of available projects comprising project ID, date and state. The state feature characterizes the client response, if any, to the commercial proposal.

3.1.2 Data to quantify first commercial proposal's lead-time

For each available project (as specified in 3.1.1), it was collected: the day the input form is uploaded to the system (PM task); the day the PD technician starts developing the commercial proposal (PD task); and the day the PD technician ends the commercial proposal development. The first two dates are automatically assigned by the system; the last one is assigned by system, contingent on the PD technician marking the task as finished. Some pre-processing was performed as inconsistencies were detected and those records were not considered – 30 records in which finish date is previous to start date; and 45 records in which start date is 01/01/1900. As noted in sub-chapter 3.3 there seems to be outliers in the data, e.g. maximum execution time is 521 days, but it was decided that no outliers should be removed, as it was not possible to understand the reason they existed.

The goal of quantifying the first commercial proposal lead time stems from the project motivation within the company – even though it was not quantified, the CD responsible thinks the team is not as responsive as it should.

3.1.3 Data to test the tool

To study the feasibility of a commercial proposal tool two distinct types of data are required: the input form (customer requirements) and the corresponding commercial proposal (identification of products that satisfy the product requirements).

The input form template is an *MS Office Excel spreadsheet* where different sheets specify different types of products. Concerning the commercial proposal, as stated in section 3.1.1, it is available in the information system database.

It was required that the input form is available in digital format (i.e. the *MS Office Excel* file), in order to allow for automatically collecting the input data. Presently there is only one PM filling in the input form in digital format; the other PM print the template and fill it in – corresponding processes have the input form available in paper format only. The commercial proposal is mainly developed directly in a tool within the information system, so the constraint is the input form. The fact that the pair input form – commercial proposal is required reduced a lot the study sample size, as only one PM fills the input form in digital format, but almost all commercial proposals are available in the information system.

A *Visual Basic for Applications* (VBA) program was developed to automatically collect the input form's data, provided the input form's file exists. 119 input forms were collected.

In a preliminary analysis of the data collected some inconsistencies were detected, which led to the deletion of 69 input forms:

- 13 input forms had not information regarding furniture or fume cupboards (target product types for the tool);
- 38 input forms had only a variable value identified – product line;
- 14 input forms did not have the corresponding commercial proposal available in the information system;
- 4 input forms requested product types (fume cupboards) but there was no product of that type in the commercial proposal, when the matching was done.

Considering the type of inconsistencies detected, these input forms were considered not valid for testing purposes, resulting in a sample size of 50 validated pairs (input form and commercial proposal).

3.1.4 Domain knowledge – variables mapping

Similarity between cases (previous projects) is not assessed by overall input form similarity but rather by input form sections (each section characterizing a different product type) – it allows the tool to identify two cases as the most similar, regarding a given product type, but not the most similar considering a different product type, e.g. project 1 is the most similar to project 2 regarding product type workbench, but considering product type cabinets, project 1 and project 3 are the most similar. Moreover, this approach enables the tool to build the solution modularly, in a similar fashion to the commercial proposal development by the PD technician.

The notation used is: there are *product types* (main product, or final product that is sold to client) that are comprised of *product families* (sub-products of the main product), e.g. the product type workbench is comprised of product family worktop, structures and connecting structures and cover panels. The product type can be seen as a modular assembly of sub-products (what the company calls product families). The input form specifies the requirements for product families.

Some domain knowledge is needed in order to understand how each product type is represented in the input form, i.e. how the variables in the input form relate to product types – a domain expert was required for this task. It was possible to arrange one meeting with a PD technician where he disclosed which variables in the input form compose each product type under analysis.

A section of the input form is presented in Figure 5. This section characterizes a product family (structures); it is annotated highlighting the variables and the values each variable can take. This section specifies a product family – namely structures, code NCZA –, which is comprised within a product type – workbenches. To fully specify a workbench different sections of the input form are required.

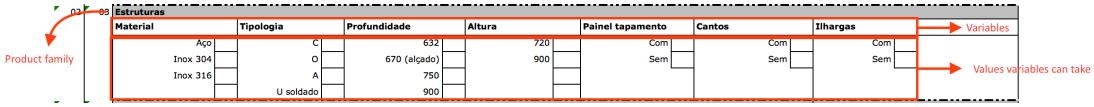

Figure 5 - Input form section characterizing product family structures

For this case, there are 7 variables, 4 categorical binary variables and 2 categorical variables, which can take 4 values, and 1 categorical variable which can take 3 unique values. These 7 variables specify a product family, and each combination of values yields a single sub-product.

In the meeting with the PD technician the knowledge of which variables specify what product families was disclosed for all product families under analysis. From the mapping variables-product family, it was disclosed the product families comprising each product type under analysis.

This is the only domain knowledge fed by an expert for the development of the tool.

3.2 Commercial proposal

Each project is assigned, at least, to one commercial proposal; on the other hand each commercial proposal is assigned to one single project.

One of the variables characterizing a commercial proposal is its state, which may assume one of five possible states – "order", "lost", "no interest", "under analysis" and "replaced":

- "Order" – the client places an order based on that commercial proposal;
- "Lost" and "no interest" – in both cases the project is not going to be pursued –the company lost the construction order to some competitor ("lost"), or the client is no longer interested in pursuing the construction ("no interest");
- "Under analysis" – the company expects feedback from the client. This is the default state assigned to any new commercial proposal;
- "Replaced" – there is, at least, one newer commercial proposal for the same project – the main reason being the client changed its specifications (the presented ones do not make sense any longer, or the client wants to add new ones). The newer proposal state has a value that is equal to one of the previous states values, or if there is more than one proposal for the same project, all but one have the state "replaced" and the remaining has a state equal to one of the previous states value.

With the exception of the "replaced" state, all possible states are meaningful in commercial terms; however, if the state is "replaced" the proposal is no longer valid – no decision will ever be taken by the client based on it (as there is a newer proposal), neither was it lost or led to an order.

In Table 1 and Table 2 statistics computed from data collected (as in 3.1.1) are presented. In Table 1 data regarding commercially meaningful states is presented, whereas in Table 2 data regarding every state is presented – Table 1 relates to the commercial aspect, Table 2 enables one to draw some conclusions about the work developed at the PD.

Table 1 - Commercial proposal characterization

Commercial proposal State	*Frequency*
Order	44,87%
Lost	5,93%
No Interest	13,76%
Under analysis	35,45%

By analysing the data with a company expert it was suggested that values regarding "no interest" are likely to be inflated and in reality distributed between "lost" and "no interest", considering it is enough the client never replied back to the commercial proposal, and there was no follow up contact, for the state to be "under analysis". It was told that statistics regarding "order" and "replaced by another commercial proposal" are probably right because the PM has to mark it "order" so the process can move forward within the company, and the "replaced" is automatically assigned by the system. An important aspect is that less than half of the developed proposals lead to an order.

Table 2 - Commercial proposals development characterization

Commercial proposal State	*Frequency*
Order	30,42%
Lost	4,02%
No interest	9,33%
Under analysis	24,03%
Replaced by another commercial proposal	32,21%

When all state values are considered it is possible to assess the weight of rework on commercial proposals development. For the year under analysis, 32,21% of the work produced by PD was somehow reworked; considering PD developing work.

3.3 First commercial proposal Lead-Time

There was the need to quantify how much time elapsed between client inquiry and the time the PM is ready to reply back, i.e. how much time does the PD takes to develop a commercial proposal.

Two types of waiting occur - queue and execution times. Queue time was not directly available and thus it was computed as the elapsed time between the day the PM uploads the form to the time the PD technician first starts developing the corresponding commercial proposal – these days are automatically assigned by the system; Execution time, too, was not directly available, and thus it was computed as the elapsed time between the day the PD technician starts the job and the day he marks it as final – marking the task as finished is done manually by the technician.

Statistics relating to the first time the PD develops a commercial proposal, for a given project, ready to send for the client (commercial conditions aside) are presented in Table 3.

Table 3 - First commercial proposal Lead-Time

	Queue [number of days]	*Execution [number of days]*
Average	6,3	1,9
Standard Deviation	12,0	15,7
Minimum	0	0
Quartile 1	1	0
Median	4	0
Quartile 3	7	0
Maximum	207	521

It is possible that the execution time data is inflated – if the PD technician forgets to mark the job as final. However, it is not expected that much time passes without anyone noticing. Concerning queue time, data correctly reflects company reality, as it is computed based on dates automatically assigned by the system.

Apparently, data presents some dispersion, considering:

- There seem to be some outliers inflating average times, as average is a little over double the median;
- The maximum values obtained clearly seem outliers, considering the average and standard deviation values – in terms of queue times, the maximum is about an average plus 17 standard deviations; in terms of execution time, the maximum is about an average plus 33 standard deviations;
- Moreover, the standard deviation seems to be high, especially if compared to average values – regarding queue times, standard is a little less than 2 times the average value, whereas relating to execution time standard deviation is a little more than 8 times the average value.

On average, between queue time and execution time, 8,2 days passed until the PM could send the first commercial proposal to the customer, and in median terms it was 4 days. Moreover, on average, 76,8% of that time was spent on queue.

3.4 Product Characterization

Considering the goal is to achieve a conceptual cost estimation of a project, product families weight in terms of return is assessed. The implicit assumption is that product families weighing more should have priority above product families weighing less in terms of return. Furthermore, as discussed in section 3.1.4, domain knowledge was acquired from a domain expert in one meeting, as such, knowing the product family relative importance also served as a guide for mapping the variables.

In a first instance it is tested if the Pareto principle is verified in the context of the company. The principle is tested considering commercial proposals for which the state is “order”, i.e. represent real return for the company, whereas if under analysis states were to be considered it would represent potential return.

To characterize the product, the data used is as described in 3.1.3. It is recalled that the sample size is 50 valid pairs of input form (in digital format) and corresponding commercial proposal.

For the sample under analysis, 97 product families are commercially proposed, of which the top 20% of families (approximately 20 families) in terms of return are analysed. In Table 4 product family's relative weight and cumulative weight are presented.

Table 4 - Product Characterization

Family	*Relative*	*Cumulative*
MBZJA	19,46%	19,46%
BAA	17,56%	37,02%
TA	16,82%	53,84%
G3	5,23%	59,08%
MBZB	3,96%	63,04%
NCZA	2,36%	65,40%
NCZB	2,32%	67,72%
25	2,11%	69,83%
AAAA	2,02%	71,84%
AABD	1,93%	73,77%
NCZC	1,76%	75,54%
JCC	1,71%	77,25%
AAAJ	1,66%	78,91%
MBZD	1,60%	80,51%
CAB	1,50%	82,00%
CDA	1,42%	83,43%
BC	1,29%	84,72%
AAAK	1,05%	85,77%
MBZC	1,02%	86,80%
CBC	0,96%	87,76%

Examining the results, it may be concluded that the Pareto principle is verified in terms of return, as approximately 20% of the families represent more than 80% of the return (87,76%).

For different reasons, some product families are out of the scope of the tool:

- MBZJA – not characterized in the input form;
- CDA – is out of the scope of the tool (not related to technical furniture or fume cupboards);
- G3 – despite being a product family related to technical furniture, it is a specialized line;
- 25 and JCC represent the ventilation system and related material that is required when a fume cupboard is installed – it requires specific development depending on the client infrastructure;

The remaining product families are, potentially, within the scope of the tool (as discussed in section 3.5.1 not all product families will be used for testing purposes).

3.5 Problem Space Characterization

In terms of the CBR system, the problem descriptions space is comprised of the input form and the problem solutions space is comprised of the commercial proposal.

The first division considered for the input form is by product main categories – technical furniture and fume cupboards – this division is not mutually exclusive, i.e., a project can require technical furniture and fume cupboards. The distribution of the projects by the product main categories is:

- 42 are technical furniture projects;
- 1 is a fume cupboard project;
- 7 are furniture and fume cupboard projects.

Despite the fact that only technical furniture and fume cupboards are being addressed, the input form is composed of several other sheets some concerning project general information, and others concerning product types that the commercial team decided should not be the target of this project.

3.5.1 The Input Form – Problem Descriptions' Space

Regarding furniture, the input form has 107 variables, whereas related to fume cupboards there are 47 characterizing variables.

Considering furniture variables, all variables are categorical. 15 variables are actually number variables, but the numbers represent categories – for instance, the variable structure depth actually takes number values, but these represent a category because the structure depth standard values are predefined into 4 categories. All the other number variables are of this type, i.e., despite taking number values, the number represents a class the variable can take. Concerning fume cupboard variables, 17 variables are categorical; the other 31 variables are numeric.

It is noted that some products are not properly defined in the input form – some features are defined by noisy variables (and should be excluded), and some features do not have any variables to characterize them – this issue is addressed in section 4.1.2.

Regarding product category, fume cupboards, there are two different product types characterized in the input form: fume cupboard and cabinets - it is noted that cabinets defined in this area are different from those defined in the technical furniture section as these are storage cabinets specialized for fume cupboards, namely, by chemical product types (reagent, acid, base, flammable substances, …). The variables in the input form allow for a complete characterization of the product type.

Concerning the other main product category, technical furniture, it is composed by several different product types, namely: workbenches, special workbenches (workbenches on castors, mobile tables, anti-vibration tables, testing booth), service systems (cells, rail, wall and gallery) cabinets (on castors, suspended, on plinth and upper units), washing modules and accessories (fittings, emergency showers and eye-washers). Among all these types, the only product type that can be completely specified in the input form is the workbench. Technical furniture specification is discussed in detail, in the following sub-sections.

To address this issue, a new input form, based on the current format, was proposed, so that at least the product families in the top 20% (which represent more than 80% of the return) are covered by it. The proposed input form is presented in sub-chapter 4.1.

Under-defined product types – the cabinets example

The case of cabinets is discussed in detail as it is at the extreme of under-specification. The company sells different types of cabinets – on castors, on plinth and suspended – and for each type there are possible variations of features such as number of drawers, number of doors, width and height.

In terms of the input form there is one variable that "defines" the required solution – that variable specifies the product type thus takes values "on castors; on plinth; suspended". No further information is provided.

To put it in context, for the case of type cabinets on castors alone there are 9 different cabinets (problem solution), by varying the combination number of doors, number of drawers, width and height. Despite the fact that these features can take different values, there are no variables in the input form representing them.

The only product type that can be specified using the current version of the input form is the workbench – all the other product types lie somewhere in between the cabinets case and the workbench case, i.e. either the product type has one variable signalling the product is required (as cabinets), or it has some additional variables but there are still missing features, which does not allow for product characterization.

Fully-defined product type – the workbench

The workbench product type can be further divided into sub-products that are defined in the input form. More precisely, each sub-product in the workbench product type has a corresponding family, as such the following sub-product and families were identified: Worktop (BAA), Structure (NCZA), Connecting Structures (NCZB), and Panels (NCZC). There are two more product families that compose the product family workbench, but they do not belong to the top 20% of families – the last family of the top 20% represents 0,96% of the return, so each of the missing families represent less than that.

Table 5 represents the problem space by family type.

Table 5 - Problem descriptions' space characterization

Family	*Number of variables*
BAA	7
NCZA	5
NCZC	2

Three clarifications have to be done:

- If the new input form is adopted (presented in the next chapter) the BAA family is characterized by 4 variables. At the moment, the number of variables is inflated by a poor definition of the variables: colour, which represents a single feature, has 4 variables to define it (this topic is further discussed in section 4.1.2).
- NCZB family is not in Table 5 because the domain expert explained no variables define it as connecting structures are standard always present in every worktop;
- The variables are not mutually exclusive – one variable can be used to characterize more than one product family.

3.5.2 The Commercial proposal – Problem Solutions' Space

Workbench is the product type that will be used to test the tool, as such in this section the problem solutions space is presented considering only this product type – as new product types are added the problem solutions space grows accordingly.

The problem solutions space is comprised of unique references, for each given family. Additional data was obtained to characterize the sample size, by product family, namely the number of unique projects in which a product family is used and the number of available commercial proposals. This data is summarized in Table 6.

Table 6 - Problem solutions' space characterization and corresponding sample size

Family	*Unique References*	*Unique Projects*	*Commercial proposals*
BAA	15	37	56
NCZA	16	33	50
NCZC	24	27	42

Regarding BAA family 15 unique products that are used in 37 unique projects represent the problem solutions space. Concerning NCZA family 16 unique products that are used in 33 unique projects represent the problem solutions space. The NCZC is the oddest case, because as explained by the domain expert there are only two variables specifying this family, but there are 24 unique references that are used in 27 unique projects.

4 Proposed Solution

In this chapter the proposed solution is presented. It is structured as follows: first the reviewed input form proposed is presented; following that, the tool developed is presented considering the developing issues of representation, similarity assessment during retrieval, case adaptation and similarity threshold influence on the adaptation result; in the end, the obtained results are presented.

4.1 Reviewed input form

The input form design does not fall directly under the scope of the project – whose aim is to study and propose a tool that automatically develops a commercial proposal. Nevertheless, considering that the input form comprises the problem descriptions space, a review of some sections is proposed to allow for the tool to cover, at least, the product families belonging to the top 20% (presented in sub-chapter 3.4) as some inconsistencies were detected when developing the tool.

The fact that the input form comprises the problem descriptions space is a design decision – it was possible that a new form was designed just for use under the developed tool. That decision would have two major drawbacks:

1. The current input form would still have to be filled in by the PM, as that is the document officially accepted within the company, and it is required to be filled in; even if the tool does not use the current input form, the tool is for PM's use whereas the input form is used in different stages of the process, by different departments (namely PD and the procurement team);
2. Extensive domain knowledge (i.e. know the product line) is required to develop such a document from scratch.

Point 1.) is the major driver in using the current input form – it is easier to integrate the tool in the process if it fits in seamlessly, i.e. no further work is required other than the push of a button. Point 2.) is not of minor importance as it is easier for a non-expert on the domain to look for what is missing, or wrong, by asking than to devise such a solution from scratch.

As stated, the current version of the input form is poorly designed in terms of product specification; if it is to be used as the problem descriptions space for a tool that is able to automatically elaborate a commercial proposal, the products have to be properly specified (see section 4.1.1). Moreover, some inconsistencies relating to noisy variables are also identified (see section 4.1.2).

4.1.1 Under-defined products

Under-defined product types pose a serious restriction to the use of a tool that is supposed to identify a product and a price based on some input requirements.

The case for cabinets was presented in section 3.5.1. This product type is said to be under-specified as if it is taken into consideration that, for cabinets on castors alone, there are 9

different cabinets (problem solution) by varying the combination number of doors, number of drawers and cabinet height. Even considering only the cabinets on castors the company sells the most – it identifies 5 – the cost range has a variation of more than 120% from the cheapest to the most expensive. It becomes clear that it is not possible to estimate cost for this product category, as there are important features that have no variables describing them.

In the same section, it is stated that all product categories lie somewhere in between the case for cabinets and the case for workbenches (which is the only product type that is properly specified in the input form) – this means that the remaining product types either have only one variable specifying some requirement (worst case scenario, as the cabinets) or there are more variables specifying different features, but not all features get defined.

The proposal regarding product families that are not specified in the input form is to update the corresponding section of the input form by adding the missing features. Developing such a proposal required the input of the PD department in understanding product features and converting it to variables and the values they take.

Following the previous example, Figure 6 shows the proposal for a new section for the case of cabinets on castors.

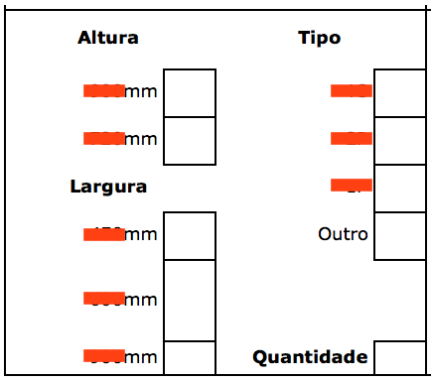

Figure 6 - Proposal to add missing features for product cabinets on castors

New variables are identified in bold – top left is a variable to specify feature height, low left is a variable to specify width and top right is a variable specifying the features number of drawers and number of doors. Low right is not a feature of the product – it represents required quantity.

Regarding the features number of drawers and number of doors, these are combined in a single feature as these usually represent a combination, e.g. 2 drawers + 1 door. Even the case when there is only one of them, e.g. 1 door, it can be seen as the combination 1 door + 0 drawers. The values this variable can take were defined by the commercial team and are considered to be the most important.

Notwithstanding only one product category is presented, this approach was extended to other product types guided by the product characterization (presented in sub-chapter 3.4) and commercial team request.

4.1.2 Noisy variables

Whereas under-defined products are an issue that might preclude the use of the tool on those product families, noisy variables, while not precluding, might seriously affect the performance of the tool.

As an example, the feature colour for the sub-product worktop, comprising product type workbench, is analysed.

There are different types of worktops and as the type represents a worktop feature there is a variable representing that feature. For the fact that not all worktop types share all the possible colours for worktops (they depend on the feature type), the designer chose to define a colour variable by worktop type (e.g., colour_for_worktop_type1, colour_for_worktop_type2, and so on) - Figure 7 is an adaptation of the corresponding section in the input form; top part represents the current situation and the bottom represents the proposed review for this section of the input form.

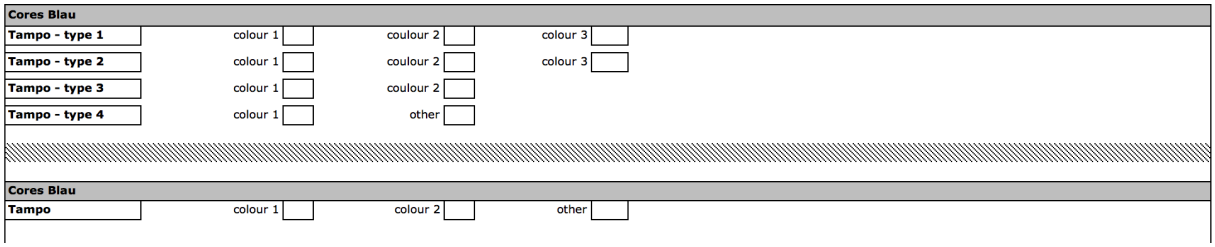

Figure 7 - Variables defining worktop colour (current and proposed situation)

Each row corresponds to a different worktop type and each column (aside from the first) represents the values the variable can take. Column 2 takes the same value for all variables (adapted to colour 1); column 3 takes the same value for the first three variables (adapted to colour 2) – the missing value for the remaining variable takes the value “other” (potentially it can take the same value as colour 2); column 4 takes the same value for the first two variables (adapted to colour 3).

Although not all worktops share all the possible colours, it does not seem reasonable defining a variable for each type, considering that all variables refer to one single feature – worktop colour. Moreover, considering the current situation, colour can be defined with one single variable taking 3 possible values and covering the current situation. The values are colour 1 and colour 2 (shared by all variables currently used) and the value “other”. “Other” covers either value 3 (for variables 1 and 2) or the last option of variable 4 in column 3 (“other”).

For worktops type 1 and type 2, colour 3 can be represented by the value “other”; for type 3 and type 4, the possible values they take in the current situation are available in the proposed situation.

There is a caveat – if the worktop type variable takes the value type 3, the PM might try to assign value “other” to the new variable colour and, in the current situation, worktop colour for worktop type 3 can only take values colour 1 or colour 2; this can be avoided by code constraining it from happening.

As presented in Table 5 (section 3.5.1) the BAA product family is currently characterized by 7 variables. If the proposed approach is adopted the product family becomes characterized by 4 variables – as the colour features becomes 1 rather than 4 variables.

4.2 Tool

In the next sections the tool developed to address the problem is described.

A tool was developed aiming to elaborate a commercial proposal based on input specifications, using CBR methodology in the reasoning process, in a construction company in the laboratories industry. Notwithstanding there are many issues that should be tackled in the development of such a tool (see sub-chapter 2.2) the main focus of this dissertation is the adaptation of cases, namely, it is a research on domain-independent strategies to overcome the knowledge engineering effort for the adaptation phase. The importance and main issues related to the adaptation phase of a CBR system were discussed in section 2.2.3 and analysed in depth in sub-chapter 2.3.

First an overview is presented, followed by the discussion of the main issues tackled: representation, similarity and adaptation.

4.2.1 Overview

The input form used in the company is a template on *MS Office Excel* and the commercial proposal is developed in a tool within the information system. It was not granted access to the central database, but the commercial proposals were made available in *MS Office Excel*. Considering this, the tool was developed in VBA.

Cases are paramount in a CBR system. In the context of the company, a case comprises the input form (client requirements) and the commercial proposal (company solution). The case-base is a collection of all the cases – as presented in section 3.1.3, the sample size is 50 validated pairs input form – commercial proposal.

The approach implemented is similar to the ones presented in (Hanney and Keane 1996, 1997; Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008) which were covered in sections 2.3.3 and 2.3.4. The researches by these authors are investigations on the case adaptation problem, namely the problem of domain independent strategies for adaptation knowledge acquisition. The differentiating aspect in this approach is the acquisition of adaptation knowledge as, aside from that, a regular CBR system is developed.

Such issues as similarity assessment and adaptation are further discussed in detail in the subsequent sections – similarity is discussed in section 4.2.3 and the details about adaptation are discussed in section 4.2.4 – however it is important to clarify some aspects before explaining how the tool works. Similarity threshold is a parameter, defined by the system developer, against which cases are considered similar during retrieval if above *similarityThreshold*. Moreover, the acquisition of adaptation knowledge is performed by pair-wise case comparison where only those pairs of cases such that similarity is above the *similarityThreshold* are compared. The importance of the parameter *similarityThreshold* stems from the fact that results from retrieval and adaptation knowledge acquisition (and thus adaptation performed) are dependent on the value to which it is set. The approach to overcome the issue of defining a meaningless value for the parameter, in the context of the problem, is to test different values for similarity threshold and opt for the one yielding the best results.

Regarding adaptation, there is one auxiliary entity used to acquire adaptation knowledge that was named adaptation case. The *adaptCase* is an object that comprises two cases (test case and retrieved case) and information relating to the comparison between them. A database of adaptation cases is created from which adaptation rules are learned.

It is now explained how the system reasons about a solution. The retrieval algorithm retrieves all the previous cases such that the similarity between cases is above the *similarityThreshold*. Retrieved cases are evaluated according to the similarity result, i.e. either the cases are an exact match (similarity = 1) or the cases are similar but have some differences (*similarityThreshold* < similarity < 1). If the similarity is equal to 1 then the test case and the retrieved case are an exact match thus no further reasoning is required and the solution of the retrieved case is proposed as solution for the test case. However if similarity is less than 1, adaptation is required to account for the differences between cases. If there are rules covering the differences then these rules are applied to the retrieved solution and, as such, the retrieved solution is adapted to account for the differences between cases; the adapted solution is proposed as the solution for the test case.

It is noted that more than one case can be retrieved. Even when similarity = 1 for different retrieved cases it is possible that retrieved cases have different solutions. In that situation, the proposed solution is the most voted solution. Likewise, it is possible that there may be more than one rule covering the differences between the test case and retrieved case. In that situation the rule with the highest *ruleFreq* is used – it denotes a measure of the confidence in the rule, based on its frequency, as in (Hanney and Keane 1996).

4.2.2 Representation

The case is represented as an object containing the problem description (input form specification) and the solution (commercial proposal). The *case* object is comprised of *ID*, *attributes, solution*; the *attributes* property contains the input form information for the case; *solution* contains the commercial proposal information for the case. The objects' properties characterization is presented in Table 7.

Table 7 – Properties of the *case* object

Property	*Data Type*	*Dimension*
caseID	String	-
attributes	Array	154
solution	2D Array	[nx3]

The *attributes* property is represented as a feature vector whose dimension is the total number of variables (as in the input form). Regarding the *solution*, it is a 2D array where n is the number of solutions in the commercial proposal, corresponding to product families under analysis; three types of information are stored – product family (column 1), product reference (column 2) and product price (column 3).

The case base is a collection object (comprised of *case* objects).

The *adaptCase* is an auxiliary entity that holds information about the comparison between two *case* objects – further details are discussed in section 4.2.4. The *adaptCase* is represented as an object comprised of *adaptcaseID*, *testCaseID*, *retrievedCaseID*, *testCaseAttributes*, *retrievedCaseAttributes*, *testCaseSolution*, *retrievedCaseSolution*, *casesSimilarity*, *productFamily* and *productType*. The objects properties characterization is presented in Table 8.

Table 8 - Properties of the *adaptCase* object

Property	*Data Type*	*Dimension*
adaptCaseID	String	-
testCaseID	String	-
retrievedCaseID	String	-
testCaseAttributes	Array	154
retrievedCaseAttributes	Array	154
testCaseSolution	2D Array	[nx3]
retrievedCaseSolution	2D Array	[nx3]
casesSimilarity	Floating-point	[0,1]
productFamily	String	-

The *adaptCase* comprises two cases, i.e. a test case and a retrieved case (*ID*, *attributes* and *solution*), and information characterizing the comparison performed (*productFamily*, and *casesSimilarity*).

As further discussed in the next section, similarity assessment is performed considering the *productFamily* and the result of comparison is a real number in the interval [0,1].

The set of *adaptCase* objects (named *adaptCases*) is a collection object which is a database holding information about all the comparisons performed.

The adaptation of a case involves adaptation rules which are created based on the *adaptCases* auxiliary database. A *rule* is represented as an object comprised of *ruleID*, *ruleConsequent*, *commonFeatures*, *diffFeatures*, *ruleFreq*, *ruleGen* and *productFamily*.

Table 9 - Properties of the *rule* object

Property	*Data Type*	*Dimension*
ruleID	String	-
ruleConsequent	2D Array	[2x2]
commonFeatures	2D Array	[ix2]
diffFeatures	2D Array	[jx3]
ruleFreq	Integer	[1,Ubound(*adaptCases*)]
ruleGen	Boolean	(1,0)
productFamily	String	-

The two-dimensional arrays *ruleConsequent*, *commonFeatures* and *diffFeatures* contain the same information, regarding adaptation knowledge, as the *adaptCase* from which the rule is generated. However, this representation allows for a different indexing of the rules which leads to more efficient retrieval of applicable rules.

The *ruleConsequent* array stores two types of information concerning the solution change – product reference change and price reference change for retrieved and target solutions.

Regarding *commonFeatures,* the index i denotes the number of features that take the same values in the *adaptCase* originating the rule; it is stored the feature ID (column 1) and the value the feature takes (column 2); Regarding *diffFeatures* the index j denotes the number of

features that take different values in the *adaptCase* originating the rule; it is stored the featureID (column 1), the value the feature takes in *adaptCase.testCaseAttributes* (column 2) and the value the feature takes in *adaptCase.retrievedCaseAttributes* (column 3). As a consequence, the sum of indices i and j equals the number of features characterizing *productFamily*.

The *ruleFreq* property denotes the rule frequency, i.e. how many times the rule is found on the dataset of adaptation cases.

4.2.3 Similarity

One parameter influencing both retrieval phase and knowledge acquisition (and thus, adaptation results) is the *similarityThreshold*. Its influence stems from the fact that retrieving the most similar case means the similarity between both cases is above *similarityThreshold*. Its influence on knowledge acquisition is that the strategy in the proposed methodology involves pair-wise case comparison of the cases, however only cases above *similarityThreshold* get to be compared.

Similarity assessment involves two different stages: setting features weights and the similarity function. The methodology used to compute feature weights is feature counting: "This method applies a weight of 1 to all the attributes, based on the understanding that unless there is concrete information on the attributes, there is no need to apply to them a weight higher than 1", (Koo et al. 2010).

Provided the weights are set, a similarity function is used to compute the similarity between cases. Similarity between cases in not assessed by bulk comparing cases, i.e. assessing similarity for all features, rather it is assessed by sections of the case (each section characterizing a different product type). The case description is comprised of input requirements on different product types, as such to assess similarity between a case and the previous cases it does not seem reasonable to assess overall similarity but instead assess similarity on the sections corresponding to the product type under consideration. Taking the example presented in section 3.1.4, this approach allows the tool to identify two cases as the most similar regarding a given product type but not the most similar considering a different product type, e.g. project 1 is the most similar to project 2 regarding product type workbench, but considering product type cabinets, project 1 and project 3 are the most similar.

The similarity function used is adapted from the approaches presented in (Doğan, Arditi, and Günaydın 2006; Koo et al. 2010). Both proposals are presented and, after that, the proposed similarity function is discussed.

The following notation has been assumed:

- TC_j, which stands for the jth feature for test case;
- $CB_{i,j}$, which stands for the jth feature for the ith case in the case base;
- FS_j, which stands for feature similarity of the jth feature.

The following conditions and blocks have been considered:

- Condition 1: TC_j is empty;
- Condition 2: $TC_j = CB_{i,j;}$
- Block 1: $FS_j = 0$ (i.e., similarity of the jth feature is 0, representing dissimilarity);
- Block 2: $FS_j = 1$ (i.e., similarity of the jth feature is 1, representing similarity).

Condition 1 is verified when the jth feature is not filled in the input form.

The proposal presented by Koo et al. (2010) is described by the algorithm:

```
Function similarityKoo (testCaseAttributes, retrievedCaseAttributes, productFamilyVariables)
    For all productFamilyVariables
        If NOT Condition 1 AND Condition 2 then
            Block 2
        Else
            Block 1
        End if
    End for
End Function
```

In practice, this similarity function evaluates similarity between cases feature if the test cases feature is not empty; otherwise it will always assign 0 similarity between cases feature (even if the ith case base element's feature j is also empty).

This raises a problem, in the sense that it might not always be true that for the fact that a feature is empty it means its similarity with other case feature is meaningless, and as such features similarity should not be evaluated, which is implicitly assumed by this similarity function.

Furthermore, if two problems are described by the same input features values (and they include meaningful empty variables) cases similarity will not be 1 (meaning, they are exactly equal) – for each empty variable this algorithm assigns feature dissimilarity, and so the overall case similarity cannot be 1.

The proposal presented by Doğan , Arditi, and Günaydın (2006) is described by the algorithm:

```
Function similarityDogan (testCaseAttributes, retrievedCaseAttributes, productFamilyVariables)
    For all productFamilyVariables
        If Condition 2 then
            Block 2
        Else
            Block 1
        End if
    End for
End function
```

In practice, this similarity function evaluates similarity between cases feature in any scenario – if the feature is empty or not.

This raises the reverse problem of Koo et al. (2010) proposal, in the sense that it might not always be true that for the fact that a feature value is empty it means its similarity with the other case's feature is meaningful, and as such similarity should be evaluated, which is implicitly assumed by this similarity function.

Furthermore, let's say that this method is used, in the absurd, to evaluate the similarity between the test case and an empty case. If there are empty variables in the test case the case's

similarity will not be 0 – for each empty variable this algorithm assigns feature similarity = 1, and so the case similarity cannot be 0, as there are some "similar" features.

Before discussing the proposed similarity function it is important to clarify what is meant by meaningfulness of a variable, when it is empty.

It is particularly critical in this specific problem because those are boolean variables describing "add-on" elements – let's consider the example of a boolean feature, that can be described as "add metallic cover panel", in this case True, or "do not add metallic cover panel", if variable is False. The fact that this variable is empty (translated as False) might not be meaningless (as would be assumed if function *similarityKoo* is used). In fact this means that the final design solution does not have a metallic cover panel as a part of the design. On the other hand, it might also happen that it is actually meaningless – if the section describing the product type is empty for all variables values, the fact that this "add-on" variable is empty is actually meaningless – it is empty because there is no element to which to add the panel.

The proposed similarity function is based on the proposal by Doğan, Arditi, and Günaydın (2006), for the fact that, in this problem context there are some variables that even empty have a meaning – and that excludes the proposal by Koo et al. (2010). However, as noted, function *similarityDogan* presents a downside, also in the context of this problem, because an empty variable is not always meaningful (as assumed in this proposal). The proposed algorithm uses function *similarityDogan* selectively:

```
Function similarityProposed (testCaseAttributes,
retrievedCaseAttributes, subProblemVariables)
     If checkProductTypeVariables then
          Call function similarityDogan
     Else
          caseSimilarity = 0
     End if
End function
```

The *checkSubProblemVariables* routine returns a boolean value denoting if the product type section comprises non-empty features, i.e. if it is true then feature evaluation must be performed as empty or not, the variable is said to be meaningful in this context.

This approach seems to make sense in the context of this problem because the meaningfulness of an empty variable, in terms of final design solution, is contingent on the values that the set of variables characterizing the same product type take – i.e. a variable that was not filled in the input form is meaningless if all variables in the same product type section are empty (checkSubProblemVariables = False) or is meaningful if there are variables that take non-empty values (checkSubProblemVariables = True).

In practice, this algorithm is an attempt to adapt the proposal in (Doğan, Arditi, and Günaydın 2006; Koo et al. 2010) to the specific needs of this problem – it neither considers empty variables as always meaningless, as in (Koo et al. 2010), nor does it consider empty variables as always meaningful (Doğan, Arditi, and Günaydın 2006). It makes a previous verification step to ensure that there are non-empty variables in the same product type section – if there are it means that any variable is meaningful and, as such, empty or not, similarity has to be assessed. If within the same product type section all variables values are empty, i.e. this product type is not required, function *similarityDogan* is not used, otherwise it could assign similarity to some variables – and in this context, the fact that the variable is empty is meaningless, and as such, similarity is not assessed.

Provided the similarityProposed function assesses features similarity, case similarity is assessed as presented in equation (1):

$$Case\ similarity\ (TC, CB_i) = \frac{\sum_{j=1}^{n} similarityProposed(TC_j, CB_{i,j}) \times w_i}{\sum_{j=1}^{n} w_j} \quad (1)$$

Where:

- TC is the test case;
- CB_i is the ith case in the case base;
- *similarityProposed* is the function proposed for feature similarity assessment;
- W_i is the features' weight array.

4.2.4 Adaptation

During the knowledge acquisition process an auxiliary database of adaptation cases is created using a CBR-retrieve only system (see Figure 4) – each adaptation case contains specific information about two similar but different cases, and how that difference reflects in the solution. Figure 8 is an example of such an adaptation case obtained by pair-wise case comparison.

Case			Problem Description					Retrieved Description					Solution		Price	
Test	Retrieved	Similarity	m_estr_material	m_estr_tipo	m_estr_prof	m_estr_alt	m_cores_estr	m_estr_material	m_estr_tipo	m_estr_prof	m_estr_alt	m_cores_estr	Retrieved	Test	Retrieved	Test
P17_00136	P17_00347	0,8	Aço	C	750	900	Branco 9010	Aço	C	750	900	Cinza L167	NCZACA0020750M00AB	NCZACA0020750M00AA	15,8	15,8

Figure 8 - Example of adaptation case obtained by pair-wise case comparison (similarity threshold = 0.7)

Having case P17_00136 compared with case P17_00347 yielded a similarity between cases of 0.8 (computed as discussed in section 4.2.3), which being above the similarity threshold is considered for comparison.

The information contained within an adaptation case (i.e. about the difference in cases and the resulting change in solution) is of type: "*IF* feature 1 (m_estr_material) equals 'Aço' *AND* feature 2 (m_estr_tipo) equals 'C' *AND* feature 3 (m_estr_prof) equals '750' *AND* feature 4 (m_estr_alt) equals '900' *AND* feature 5 (*m_cores_estr*) changes from value 'Branco 9010' to 'Cinza L167' THEN Change *retrieved solution* from NCZA(…)A to NCZA(…)B *AND* add 0 to *retrieved price*".

Adaptation rules are created based on the dataset of adaptation cases but differ in the structure; adaptation knowledge contained in a rule is structured in 3 two-dimensional arrays: *commonFeatures*, *diffFeatures* and *ruleConsequent*. Figure 9 presents the rule generated from the adaptation case example presented in Figure 8.

Common Features					Different Features										Solution Change				
5	6	7	8	100	5	5	6	6	7	7	8	8	100	100					
m_estr_mat	m_estr_tip	m_estr_pr	m_estr_alt	m_cores_e	m_estr_ma	m_estr_ma	m_estr_tip	m_estr_tip	m_estr_pr	m_estr_pr	m_estr_alt	m_estr_alt	m_cores_e	m_cores_e	Retrieved	Test	Retrieved	Test	fRUle
Aço	C	750	900										Branco 901	Cinza L167	NCZACA0020750M00AB	NCZACA0020750M00AA	15,8	15,8	2

Figure 9 – Example of Adaptation Rule (similarity threshold = 0.7)

The information contained within the rule is of type: *IF* the set *commonFeatures* is [5,6,7,8] *AND commonFeatures* takes values ['Aço', 'C', '750', '900'] *AND* the set *diffFeatures* is [100] *AND diffFeatures* changes take values ['Branco 9010', 'Cinza L167'] *THEN* change *retrieved solution* from NCZA(…)A to NCZA(…)B *AND* add 0 to *retrieved price*".

Whereas in terms of information regarding adaptation it is the same as the information contained in an adaptation case, this representation allows for a different indexing of the rules, which leads to more efficient retrieval of applicable rules. The *fRule* column in Figure 9 represents the property *ruleFreq* which denotes the rule frequency, i.e. how many times is the

rule found on the dataset of adaptation cases – for this example, this rule is found in 2 different adaptation cases.

Notwithstanding the similarity importance, it deeply affects the rule generation process and the type of adaptation knowledge acquired. The fact that two cases are less similar (a lower similarity threshold is parameterized) does not imply that the knowledge acquired is less valuable (as presented in sub-chapter 4.3).

By comparison of the example presented in Figure 9, where a rule generated with a similarity threshold equal to 0.7 is presented, a rule generated with a similarity threshold of 0.5 is presented in Figure 10.

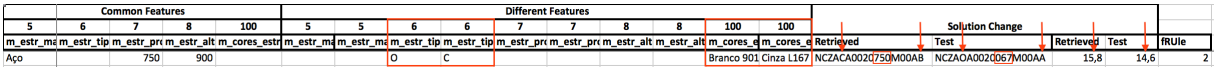

Figure 10 – Example of Adaptation Rule (similarity threshold = 0.5)

With less similar cases there can be rules generated handling more differences, which may be more valuable to the system.

Regarding adaptation rules application it is processed as follows: when the similarity between the test case and the retrieved cases is less than one, the system assesses the common and different features between the test case and the retrieved case and searches the rules library for rules that satisfy the feature differences identified. Rule search is performed by looking for rules that satisfy *diffFeatures* first, and then *commonFeatures* – the implicit assumption is that if cases are similar it is more likely that there are more *commonFeatures* than *diffFeatures*, thus it is easier to search the adaptation cases database by rules covering *diffFeatures*.

Provided there is at least one rule covering the *diffFeatures*, it needs to ensure that the rule is applicable to the problem, i.e. *commonFeatures* also match. As an example, considering the system is looking for how to adapt a solution with worktop width = 632, but the test case specifies worktopWidth = 750. There may be rules covering this *diffFeatures*, however it is needed to ensure the rest of the problem holds: for instance, the system may have generated rules for a given worktop material (e.g. worktop = "compact") but has never seen width changes for a different material type (e.g. worktop = "labgrade") – a width change within a given material type might not be applicable for different material types; an example of two rules covering the same *diffFeatures* set but with different *commonFeatures* set is presented in Figure 11.

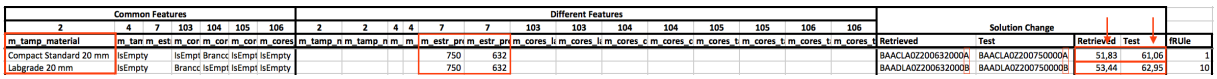

Figure 11 - Rules comparison for equal *diffFeatures* and different *commonFeatures*

For the same *diffFeature* the solution change is different as the *commonFeatures* differs, i.e. the rest of the problem description does not hold. As will be discussed in sub-chapter 4.3 this restriction was dropped as it was too restrictive given the space coverage available; nevertheless, it is important to notice that ideally such a constraint violation should be avoided.

It is noted that the adaptation knowledge acquired by this methodology, of which rules presented in Figure 9, Figure 10 and Figure 11 are examples, is achieved without domain knowledge fed in the system – using Wilke et al. (1997) terminology, it is a knowledge-light

approach for transferring knowledge (already acquired) from the case-base container to the adaptation container, after transformation.

Adaptation metrics generated using this method are presented in Appendix E.

4.3 Results and Discussion

The only domain knowledge used in this tool is as presented in section 3.1.4, i.e. mapping the variables in the input form to the product families for that corresponding section – aside from that, all knowledge is acquired as described in the previous chapters. To assess the knowledge acquisition process it is compared with results from the retrieval system, i.e. as the adaptation occurs after retrieval it is assessed if there is an improvement.

The tool is tested using cross-validation, i.e. for all cases in the sample each case is set as test case and the remaining are used as training data. The program is run and the proposed solution is assessed against the case solution, which is known.

The problem addressed the possibility of elaborating a commercial proposal, which requires correctly identifying the product that best meets the client needs. In order to measure this capability the measure used for evaluation is *accuracy* on product identification. Considering the purpose, maybe even more important than correctly identifying the product reference, it is important to evaluate the proposed price – as presented in sub-chapter 1.3, the CD required that the proposed solution cost and the actual solution cost are within the range of ± 20%. Being over the actual cost means the CD may be losing competitiveness by presenting unnecessarily expensive solutions, whereas being under the actual cost means the CD must assume the error, thus losing on commercial margin or, in the worst case scenario, incurring in cost. The proposed metrics to assess the CD requirements is *mean percentage error* (MPE) *mean absolute percentage error* (MAPE), as well as the *percentage of proposals within the required range*.

Considering only MAPE or MPE does not allow the analysis on the tool proposals in terms of proposing costs over, or under, the actual product cost – MAPE provides an overview on the tool cost identification capabilities but ignores proposals under the actual cost (negative results); on the other hand, by not using absolute values MPE weighs both under and over cost proposals which might hide problems as negative results may cancel positive ones. Given that, it is argued that to evaluate cost proposals both MAPE and MPE should be considered.

The results presented are those of the models considered to yield the best results – in Appendix A through Appendix D the detailed results for each similarity threshold used are presented.

The results are reported with the corresponding similarity threshold for which it was obtained between parentheses. The tool was tested with the initial sample size of 50 for product type workbenches (which comprises product families BAA, NCZA and NCZC). In Table 10 and Table 11 the results for the retrieve-only and after adaptation, respectively, are presented.

Table 10 – Best tool results for retrieve-only

Product Family	*Accuracy*	*MAPE*	*MPE*	*Percentage within ±20%*
BAA	39% (0.7,0.5,0.4)	13,0%(0.8)	5,3%(0.5,0.4)	83% (0.8)
NCZA	44% (0.5)	10,6%(0.4)	4,6% (0.8)	79% (0.5)
NCZC	18% (0.5)	53,0%(0.5)	40,6%(0.7)	31% (0.5)

Table 11 – Best tool results after adaptation

Product Family	*Accuracy*	*MAPE*	*MPE*	*Percentage within ±20%*
BAA	36%(0.5, 0.4)	13,5%(0.8)	-3.6%(0.5)	83% (0.8)
NCZA	46%(0.5)	4,03%(0.4)	4,0%(0.7)	77 %(0.4)
NCZC	-	-	-	-

It is noted, as presented in Table 6, there are 15 possible solutions for BAA family, 16 possible solutions for NCZA family and 24 for NCZC product family.

The first conclusion to be drawn is that applying the acquired adaptation knowledge does not seem to have a significant impact on overall results.

The worst overall results are obtained for product family NCZC. If one has to draw some possible explanation it would be that this product category is also under-defined, but it somehow was not clarified in the meeting with the PD technician. This hypothesis seems to make sense if it is considered that as presented in Table 5, two variables were identified as specifying the NCZC product family, and, according to Table 6, there are 24 possible solutions. Considering that for each of the 2 variables they can take 2 distinct values each, it yields 4 possible combinations. Even considering that more than one solution can be specified by the same requirements there are 6 times more possible solutions than possible input requirements. As it was only possible to hold one meeting with a PD technician it was not possible to confirm if this is actually the case, or if there is some other reason. In case it is in fact properly specified some domain knowledge is required for the retrieval system.

Furthermore, there are no adaptation results for product family NCZC which means that either the system does not ask for adaptation – it is recalled that, as explained in section 4.2.1, the system asks for adaptation if it retrieves cases with similarity under 1, otherwise, as cases match, the system does not ask for solution adaptation and uses the most voted of the retrieved solutions – or the adaptation mechanism was not able to generate rules covering the differences in the retrieved cases. In the case of product family NCZC the system always retrieves cases with similarity equal to 1, so adaptation is never asked.

The best results are obtained for product family NCZA – overall results for accuracy, MAPE and MPE are improved, although marginally, aside from MAPE which is cut in a little over half (from 10,6% to 4,3%). As for the metric percentage within ±20% it is marginally worse after adaptation.

As for product family BAA the results are generally worse, however, as for product family NCZA the overall impact seems to be marginal.

On the upside, considering the commercial challenge, it is noted that for product families BAA and NCZA both MAPE and MPE are within the required range the CD asked, i.e. under ±20% error on cost. The same does not hold for product family NCZC; it is argued, as previously discussed, that there might be some specification problem which does not allow for proper retrieval – it is recalled that even though the overall results are poor the system never asked for adaptation, i.e. retrieves exactly matching cases.

The main motivation for exploring different *similarityThreshold* parameter values is as de Mántaras et al. (2005) point “several authors have questioned the basic assumption on which similarity-based retrieval is based, namely that the most similar cases are the most useful for

solving the target problem", which seems to hold for this case too. Different similarity thresholds were tested (from 0.4 to 0.8) and the best results of adaptation in terms of accuracy, MAPE and percentage within ± 20% for NCZA are obtained for similarities 0.5 and 0.4, which strengthens Leake (1996) and de Mántaras et al. (2005) claim that it has to be taken into consideration the notion of most *usefully* similar case rather than the most similar case. It is noted that the best result for MPE was obtained for similarity threshold 0.7 however, not by much (see Appendix C and Appendix D).

As for family BAA, it seems to conflict with the notion of usefully similar, and rather seems to be true that only the most similar needs to be considered – it is argued it is not the case. It is believed it only occurred because, as argued in section 4.1.2, variables for family BAA namely colour, are poorly defined. Allowing lower similarity thresholds implies allowing more features to be different. However, given that there are noisy variables, more noise is allowed and the knowledge acquisition mechanism ends up acquiring poor rules. If that holds, then it seems reasonable that only high similarities should be considered for this case as a way of controlling allowed noise within the system – in this case, noise remains in the case-base container, rather than being also transferred and transformed within the adaptation container.

Notwithstanding these results, they provide the tool output results but they do not allow for a proper assessment on the adaptation mechanism designed – i.e., the results present the overall results by the system considered it uses retrieval-only or both retrieval and adaptation. However, as explained previously, the adaptation mechanism is used by request if the system does not retrieve exactly matching cases, but rather it retrieves similar cases that require adaptation.

The results of the adaptation mechanism only are presented in Table 12 (for product family BAA) and Table 13 (for product family NCZA).

Table 12 - Adaptation results for product family BAA

similarityThreshold	*Accuracy*	*MAPE*	*MPE*
0.8	0	26,1%	-12,3%
0.7	0	32,8%	5,4%
0.5	0	32,8%	5,4%
0.4	0	32,8%	5,4%

Table 13 - Adaptation results for product family NCZA

similarityThreshold	*Accuracy*	*MAPE*	*MPE*
0.8	67%	0,0%	0,0%
0.7	67%	0,0%	0,0%
0.5	67%	2,7%	2,7%
0.4	67%	1,6%	-1,6%

These results are also mixed, depending on the product family considered. As for product family BAA the results are really poor – results are worse than retrieve only mechanism, which indicates that if the system asks for adaptation for product family BAA it is expected

that the proposed adapted solution is even worse; this was expected considering that, as previously stated, adapted results for product family BAA are marginally worse.

If product family NCZA is considered, it seems reasonable to conclude that these results appear to be encouraging, at least from the adaptation mechanism standpoint. When the system asks for an adaptation, the adaptation mechanism can correctly identify 67% of the products. Moreover, for the best results obtained, even when missing the correct product reference, the adaptation mechanism proposes solutions that are very close in terms of cost – MAPE and MPE are equal to 0% for similarity thresholds 0.8 and 0.7, and within the range ±3% for similarity thresholds 0.5 and 0.4.

A possible explanation for the fact that the results for product families BAA and NCZA are so different, in terms of adaptation, is, as argued in the case of *similarityThreshold*, variables for family BAA, namely colour, are poorly defined and might lead the knowledge acquisition mechanism to acquire poor rules.

The number of rules generated by *similarityThreshold* parameter values is presented in Appendix E. Notwithstanding that considerable more adaptation rules are generated the lower the *similarityThreshold* is, it does not translate in quality adaptation rules. This seems to be the case for product family BAA, especially, as even though lower *similarityThresholds* are tested it does not translate to better adaptation results.

5 Conclusion and Future Work

In this chapter the main conclusions drawn from the project are presented (sub-chapter 5.1). Next, some limitations are acknowledged and possible paths of improvement are pointed out (sub-chapter 5.2).

5.1 Conclusions

Results obtained seem to be promising to attain the CD goal. In terms of product families BAA and NCZA the metric that best reflects the CD request, MAPE, is within the requirements 13,0 % and 10,6% respectively – clearly within ± 20 % interval. In terms of accuracy, which would result, as a side effect, in savings of time for the PD technicians, it is around 39% for BAA family and 46% NCZA, which does not seem acceptable. As for product family NCZC results are very poor for whichever metric is considered.

Regarding the adaptation knowledge acquisition method investigated, it seems to be adequate for the task at hand. Results obtained for proposed solutions after adaptation, for product family NCZA seem to handle the adaptation task as supposed. These seem to be good results, especially if one considers that the system is able to adapt a solution (from the most voted of the most similar cases) with no domain knowledge about adaptation being fed into it.

It is argued that for different reasons adaptation failed for the other product families under analysis – for product family BAA it is argued that poor definition of variables misleads the adaptation mechanism leading it to acquire poor adaptation knowledge; in the case of product family NCZC there are no results for the fact that the retrieval mechanism does not acknowledge the need for adaptation. If it holds, two different approaches address these issues:

1. Redesigning the input form, as described in section 4.1.2, to eliminate noisy variables, which mislead both the retrieval and the adaptation mechanisms – tackles the problem raised in the product family BAA;
2. Redesigning the retrieval mechanism – this is further discussed in sub-chapter 5.2, as it is a problem to be addressed; clearly that is the issue in the product family NCZC, as the system is not able to recognize the need for adaptation.

Considering the results presented in Appendix E, it clarifies that the retrieval mechanism must be reviewed. The system requires adaptation for the minority of cases, despite the fact that considering the results presented in 4.3 results can be improved, especially in terms of accuracy on product reference identification.

Moreover it is argued that the case-base size might have been an issue, as it does not provide sufficient problem coverage. The adaptation mechanism cannot always perform proper adaptation (see appendix E), which points out that despite acquiring knowledge it is unable to adapt a solution. If the case-base were bigger and obtained by random sampling it would yield a higher confidence.

Notwithstanding, considering the methodology applied, it is expected the system is able to improve along time. As Leake (1996) points out, "CBR systems do incremental learning, they can be deployed with only a limited set of 'seed cases', to be augmented with new cases if (and only if) the initial case library turns out to be insufficient in practice". In fact, these results are obtained with a sample size of 50 cases and maybe with additional cases, more coverage is provided and more refined knowledge can be acquired. Furthermore, "CBR offers a significant benefit for knowledge maintenance: a user may be able to add missing cases to the case library without expert intervention", (Leake 1996). Considering the context presented, it is, in fact, easy to update the knowledge base as it is, at the moment, an Excel worksheet – one spreadsheet contains the exported commercial proposals (can be updated the same way), and the input form data is automatically collected, provided the input forms are in a given folder.

Similarity seems to be a crucial issue in a CBR system. In this specific case, where adaptation knowledge is not implemented and rather acquired, thus similarity also affects the adaptation phase, it seems to be even more important. Despite the fact that it is not clear from the results, it is argued that similarity within a CBR system should be viewed from the notion of usefully similar rather than the most similar. Considering what is argued, it is expected this conclusion would be crystal clear if not for noisy variables defining the BAA product family.

It is expected that if the input form proposal (product families properly specified) is adopted and it is possible to collect a representative sample the CD can implement the developed tool within its process.

Aside from the company issues, it is argued that the methodology for adaptation knowledge acquisition proposed by Hanney and Keane (1996, 1997) on which this tool is based is suitable for acquiring adaptation knowledge in a CBR tool. This knowledge-light approach to adaptation rule learning is an attempt to "improve on the performance of a retrieval system without costly knowledge engineering" (Hanney and Keane 1996, 1997). In this specific application it seems possible to use such an approach, although further studies are required as new cases are acquired.

5.2 Future Work

The tool presented has some limitations, which are now pointed out.

In the retrieval system, the feature setting method used – feature counting – implicitly assumes that features have equal importance, which is a rough assumption. There are some researches which might guide in improvements in this area – in (Doğan, Arditi, and Günaydın 2006; Koo et al. 2010) the authors investigated the use of an evolutionary approach (using genetic algorithms) in the feature setting and managed to improve the retrieval results.

A different structure to case indexing might be useful especially for large case-bases, as "when the case memory is large, a good organization of the memory is a must because a simple linear organization, like a list, is very inefficient for retrieval. A hierarchical organization is necessary", (de Mántaras and Plaza 1997). In this project, as the case-base is small, such issues are not dealt but, in case it is implemented, it can soon become an issue. In (Wess, Althoff, and Derwand 1993) an approach to case representation is presented, where cases are represented in a tree-like structure – this is a possible path to explore.

Jarmulak, Craw and Rowe (2000) explore the use of genetic algorithms to improve case indexing and retrieval in a similar tree-structure – this approach combines the suggestions of both previous paragraphs – use of an evolutionary approach to optimise feature weighting and the use of a tree-structure as case representation.

Concerning case representation, the work by Cunningham and Bonzano (1999) points out a strategy in addressing the knowledge engineering issues in developing a CBR system, namely case representation. The authors point out that discussion with domain experts helped clarifying the appropriate case representation, namely by analysing retrieval results with a domain expert and discussing the reasons the retrieved cases are or are not adequate. Three iterations were performed to fine-tune case representation.

Last, there are some works (Wiratunga, Craw, and Rowe 2002; Craw, Wiratunga, and Rowe 2006; Policastro, Carvalho, and Delbem 2006, 2008) based on the methodology proposed in (Hanney and Keane 1996, 1997). In these works the authors try to refine the adaptation-knowledge acquired by exploring different machine learning techniques to learn and generalize the knowledge in the adaptation dataset. It was not possible to explore this path to rule generalization but it seems to be an approach to explore. In their works the authors showed the use of such techniques (different techniques are explored in different domains) can improve the adaptation knowledge acquired.

References

Aamodt, Agnar, and Enric Plaza. 1994. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches." *AI Communications* 7 (1). IOS Press: 39–59. doi:10.3233/AIC-1994-7104.

Bergmann, Ralph, Janet Kolodner, and Enric Plaza. 2005. "Representation in Case-Based Reasoning." *Knowledge Engineering Review*. Cambridge University Press. doi:10.1017/S0269888906000555.

Craw, Susan, Nirmalie Wiratunga, and Ray C Rowe. 2006. "Learning Adaptation Knowledge to Improve Case-Based Reasoning." *Artificial Intelligence* 170: 1175–92. doi:10.1016/j.artint.2006.09.001.

Cunningham, P., and A. Bonzano. 1999. "Knowledge Engineering Issues in Developing a Case-Based Reasoning Application." *Knowledge-Based Systems* 12 (7): 371–79. doi:10.1016/S0950-7051(99)00042-8.

Doğan, Sevgi Zeynep, David Arditi, and H. Murat Günaydın. 2006. "Determining Attribute Weights in a CBR Model for Early Cost Prediction of Structural Systems." *Journal of Construction Engineering and Management* 132 (10): 1092–98. doi:10.1061/(ASCE)0733-9364(2006)132:10(1092).

Hanney, Kathleen, and Mark T Keane. 1997. "The Adaptation Knowledge Bottleneck: How to Ease It by Learning from Cases." In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1266:359–70. doi:10.1007/3-540-63233-6_506.

Hanney, Kathleen, and Mark T. Keane. 1996. "Learning Adaptation Rules from a Case-Base." In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1168:179–92. doi:10.1007/BFb0020610.

Jarmulak, Jacek, Susan Craw, and Ray Rowe. 2000. "Genetic Algorithms to Optimise CBR Retrieval." In *European Workshop on Advances in Case-Based Reasoning EWCBR 2000: Advances in Case-Based Reasoning*, 136–47. Springer, Berlin, Heidelberg. doi:10.1007/3-540-44527-7_13.

Kolodner, Janet L. 1992. "An Introduction to Case-Based Reasoning." *Artificial Intelligence Review* 6 (1): 3–34. doi:10.1007/BF00155578.

Koo, ChoongWan, TaeHoon Hong, ChangTaek Hyun, and KyoJin Koo. 2010. "A CBR-Based Hybrid Model for Predicting a Construction Duration and Cost Based on Project Characteristics in Multi-Family Housing Projects." *Canadian Journal of Civil Engineering* 37 (5): 739–52. doi:10.1139/L10-007.

Laborial Soluções para Laboratório, SA. 2016. "Apresentação." https://www.laborial.com/apresentacao.html.

Leake, David B., ed. 1996. “CBR in Context: The Present and Future.” In *Case-Based Reasoning : Experiences, Lessons and Future Directions*, 3–30. AAAI Press. http://www.aaai.org/Press/Books/leake.php.

Mántaras, Ramón López de, David Mcsherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, et al. 2005. “Retrieval, Reuse, Revision and Retention in Case-Based Reasoning.” *Knowledge Engineering Review*. Cambridge University Press. doi:10.1017/S0269888906000646.

Mántaras, Ramón López de, and Enric Plaza. 1997. “Case-Based Reasoning: An Overview.” *AI Communications* 10: 21–29.

Mitra, Rudradeb, and Jayanta Basak. 2005. “Methods of Case Adaptation: A Survey.” *International Journal of Intelligent Systems* 20 (6): 627–45. doi:10.1002/int.20087.

Policastro, Claudio A., Andre C.P.L.F. Carvalho, and Alexandre C.B. Delbem. 2006. “Automatic Knowledge Learning and Case Adaptation with a Hybrid Committee Approach.” *Journal of Applied Logic* 4 (1): 26–38. doi:10.1016/j.jal.2004.12.002.

Policastro, Claudio A., André C.P.L.F. Carvalho, and Alexandre C.B. Delbem. 2008. “A Hybrid Case Adaptation Approach for Case-Based Reasoning.” *Applied Intelligence* 28 (2). Springer US: 101–19. doi:10.1007/s10489-007-0044-4.

Richter, Michael M., and Agnar Aamodt. 2005. “Case-Based Reasoning Foundations.” *The Knowledge Engineering Review* 20 (3). Cambridge University Press: 203. doi:10.1017/S0269888906000695.

Smyth, Barry, and Mark T. Keane. 1998. “Adaptation-Guided Retrieval: Questioning the Similarity Assumption in Reasoning.” *Artificial Intelligence* 102 (2): 249–93. doi:10.1016/S0004-3702(98)00059-9.

Watson, I. 1999. “Case-Based Reasoning Is a Methodology Not a Technology.” *Knowledge-Based Systems* 12 (5–6). Elsevier: 303–8. doi:10.1016/S0950-7051(99)00020-9.

Wess, Stefan, Klaus-Dieter Althoff, and Guido Derwand. 1993. “Using K-D Trees to Improve the Retrieval Step in Case-Based Reasoning.” *First European Workshop, EWCBR-93 Kaiserslautern, Germany, November 1–5, 1993 Selected Papers*, 167–81. doi:10.1007/3-540-58330-0_85.

Wilke, Wolfgang, Ivo Vollrath, Klaus-Dieter Althoff, and Ralph Bergmann. 1997. “A Framework for Learning Adaptation Knowledge Based on Knowledge Light Approaches.” In *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, 235–42.

Wiratunga, Nirmalie, Susan Craw, and Ray Rowe. 2002. “Learning to Adapt for Case-Based Design.” *Advances in Case-Based Reasoning* 2416: 421–35. https://link.springer.com/content/pdf/10.1007/3-540-46119-1_31.pdf.

Appendix A: Results for similarity threshold = 0.8

Retrieval Only					
Product Family	**N cases**	**Accuracy**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	46	37%	**13,0%**	-5,7%	**83%**
NCZA	32	41%	11,5%	4,6%	66%
NCZC	49	16%	53,6%	40,6%	24%
NCZH	18	33%	23,1%	12,5%	67%
Adaptation Results					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	46	35%	13,5%	-6,3%	83%
NCZA	32	44%	10,4%	**4,0%**	69%
NCZC	0	-	-	-	-
NCZH	0	-	-	-	-

Appendix B: Results for similarity threshold = 0.7

Retrieval Only					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	54	**39%**	13,8%	-5,7%	80%
NCZA	32	41%	11,5%	4,6%	66%
NCZC	49	16%	53,6%	40,6%	24%
NCZH	18	33%	23,1%	12,5%	67%
Adaptation Results					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	54	35%	15,9%	-5,0%	80%
NCZA	64	44%	10,4%	4,0%	69%
NCZC	0	-	-	-	-
NCZH	0	-	-	-	-

Appendix C: Results for similarity threshold = 0.5

Retrieval Only					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	56	**39%**	13,5%	-5,3%	77%
NCZA	39	44%	11,0%	5,4%	**79%**
NCZC	62	**18%**	53,0%	42,1%	**31%**
NCZH	0	-	-	-	-
Adaptation Results					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	56	36%	15,5%	**-3,6%**	79%
NCZA	39	**46%**	9,8%	4,2%	74%
NCZC	0	-	-	-	-
NCZH	0	-	-	-	-

Appendix D: Results for similarity threshold = 0.4

Retrieval Only					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	56	**39%**	13,5%	-5,3%	79%
NCZA	39	38%	10,6%	6,1%	72%
NCZC	0	-	-	-	-
NCZH	0	-	-	-	-
Adaptation Results					
Product Family	**N cases**	**Accuracy (Class)**	**MAPE**	**MPE**	**< Thresh. Err. 0,2**
BAA	56	36%	15,5%	**-3,6%**	79%
NCZA	39	44%	**9,2%**	4,4%	77%
NCZC	0	-	-	-	-
NCZH	0	-	-	-	-

Appendix E: Adaptation measures by *similarityTheshold* paramteter

similarityThreshold	*0.8*	*0.7*	*0.5*	*0.4*
Number of *adaptationCases*	846	1058	1721	2122
Average Number of *rules*	240	499	1252	1599
Number of Adaptation Requests	14	18	29	25
Number of Adaptations Performed	7	11	11	11